CubeSat Flight Software Workshop

Flight Software Design

Michael Starch
Flight Software Engineer
June 4, 2019

Jet Propulsion Laboratory
California Institute of Technology

© 2019 California Institute of Technology. Government sponsorship acknowledged.

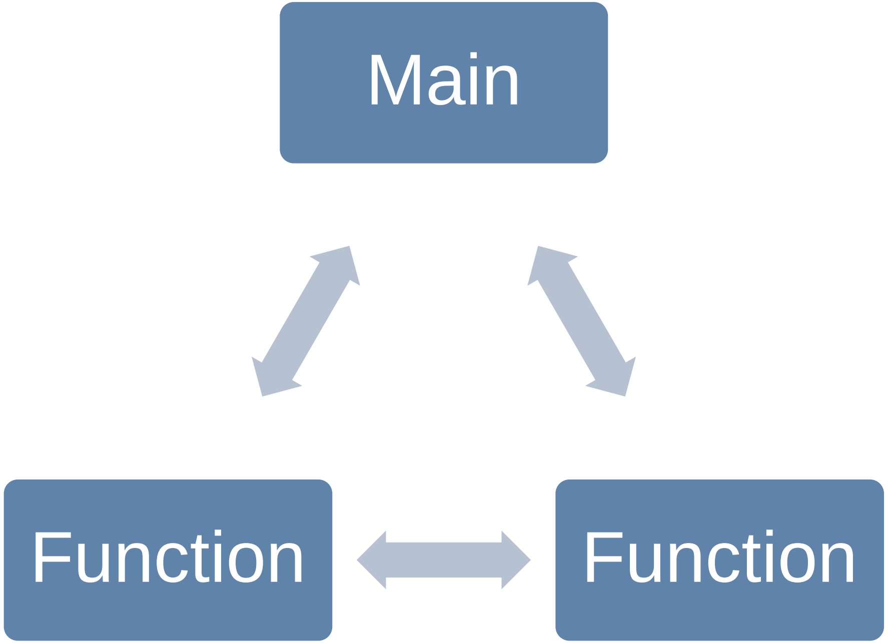
Overview of this Lesson

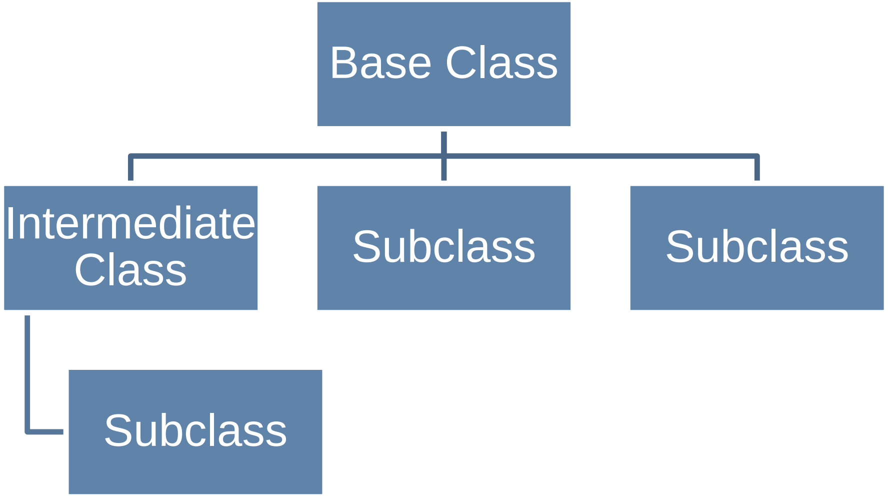
- Software Systems Design
- Systems Breakdown
  - Functionality
  - Interfaces
  - Data and Data Path
  - Off-Nominal Conditions
- Design Considerations
  - Initialization and Allocation
  - Deadlines, Timeliness
  - Concurrency, Threads
  - Faults, FATALs, and Error Handling
- Modeling Flight Software in F´
  - Topologies, Components, and Ports
  - Data Serialization
- F´ Built-Ins
  - Fw: Core Types
  - Os: Operating System Abstraction
  - Svc: High-level service components
- F´ Standard Patterns
  - Adapter
  - Manager - Worker
  - Rate-groups Timeliness
  - Hub Pattern

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Software Systems Design

Software as a System (Static)





© 2019 California Institute of Technology. Government sponsorship acknowledged.

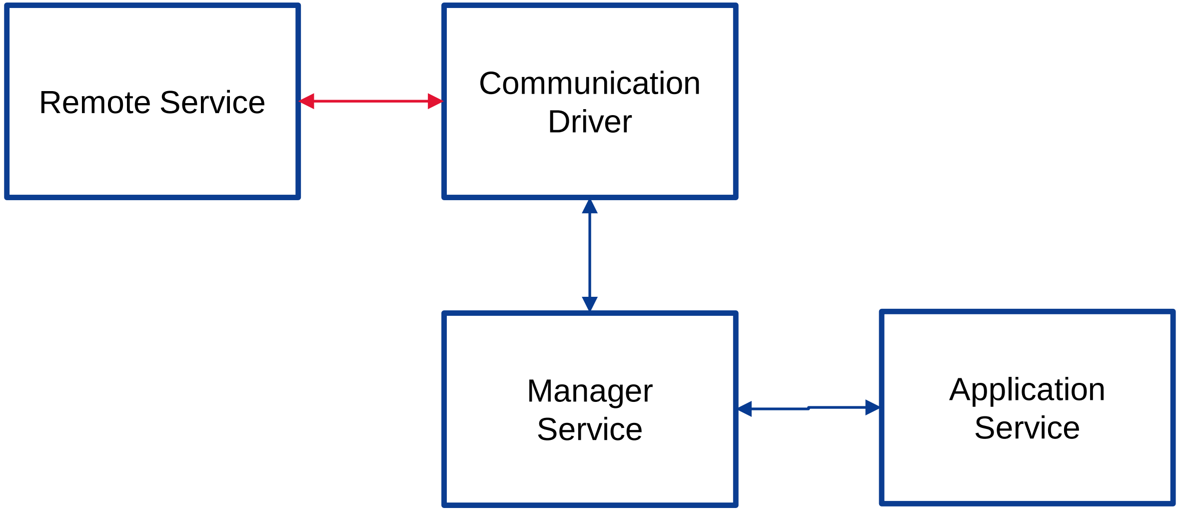
Software as a System (Static)

```
/**
 * Table of contents approach
 */
int main(int argc, char** argv) {
    int output = step1();
    step2(output);
}
```

```
/**
 * Interacting services
 */
int main(int argc, char** argv) {
    MyService service1;
    OtherService service2;
    service1.register(service2);
}
```

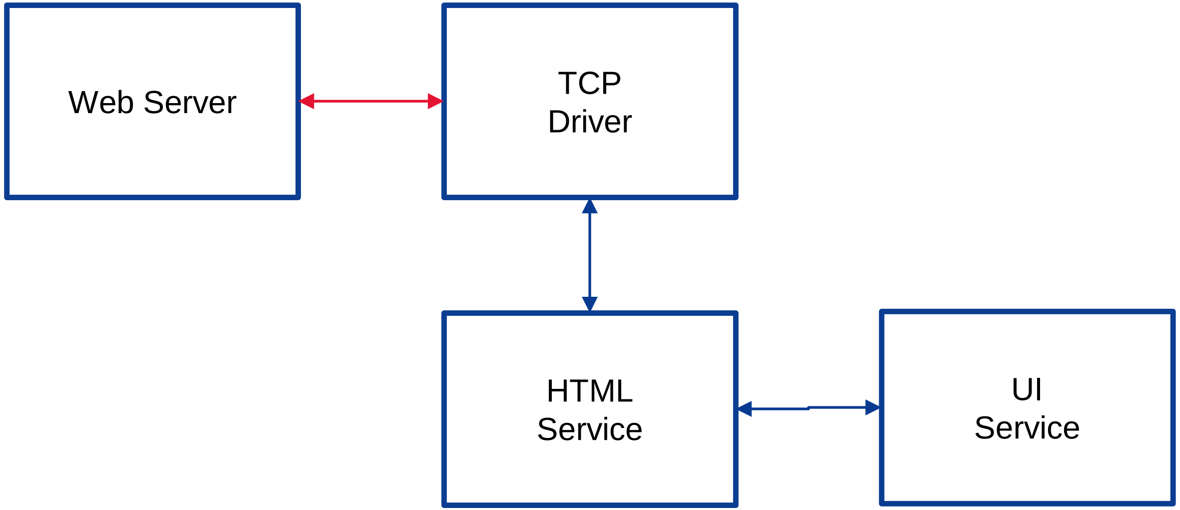
© 2019 California Institute of Technology. Government sponsorship acknowledged.

Software as a System: Systems Design



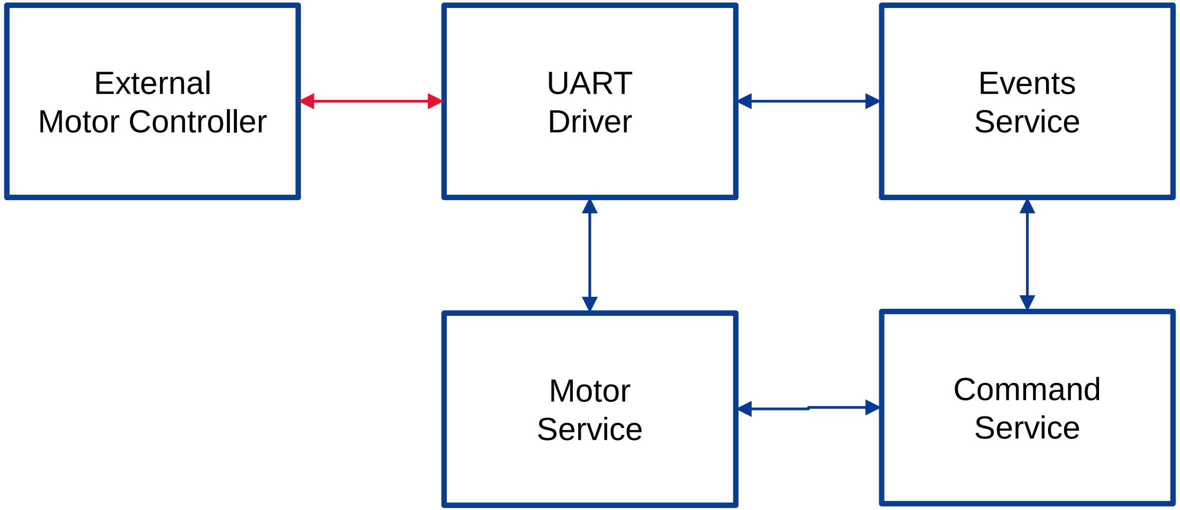
© 2019 California Institute of Technology. Government sponsorship acknowledged.

Software as a System: Web Browser Example



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Software as a System: Embedded Example



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Systems Breakdown

System Functionality

- Identify system components
- Identify aspects requiring software support
- List software components
- Identify needed interaction for interface planning

- Examples:
  - Motor controller -> software motor manager
  - Motor controller -> hardware driver (UART, SPI, I2C…)
  - Radio communication -> radio component manager
  - Radio communication -> radio hardware driver (SPI, etc…)

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Interface Design

- Interfaces Specify Interactions between components:
  - Protocol: language used for communication (Function calls, Register Writes, Packets)
  - Exposed functionality: functionality available for other components
- Identify data types and data paths in the system
- Examples:
  - Event manager sends packetized events to radio manager via function call
  - Radio manger sends packets via a function call to SPI driver
  - SPI driver writes registers via SPI to send data our radio

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Data Types and Data Paths

- Identify what data in the system
- Identify the type of the data and necessary serialization
- Identify the owner of data items at each stage
- Dynamic items allocated from static pool, failures handled

- Booleans and flags
- Bytes and byte arrays
- Integers and fixed-widths
- Floating points types
- Arrayed types
- Data structures

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Off-Nominal Conditions

- Identify places where non-standard conditions can occur
- Identify the severity of the condition
- Identify appropriate response to the condition

- Examples:
    - Hardware failure -> go to safe mode
    - Malformed user input or data -> emit warning event
    - Memory inconsistency -> full system reset

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Flight Software Design Considerations

Startup: Initialization and Allocation

- Finite resources are allocated at initialization to reduce risk
- Typical resources include: RAM, Threads, Critical Files
- Dynamic resources draw from preallocated pool; failures handled

- Implications:
  - Static memory or initialization allocated heap
  - Preallocated worker threads
  - Preallocated critical files
  - No recursion
  - Buffer managers, file managers handle unpredictable requests

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Deadlines, Timeliness

- Some processes have strict timing or deadlines
- Identify tasks that require strict timing, active processing, and background processing
- Strict timing needs specific deadline handling

- Implications:
  - Non-time sensitive work should be placed in low-priority background tasks
  - Critical work without deadlines occupies medium priority tasks
  - Work with specific deadlines goes in the highest priority tasks

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Synchronous Execution, Concurrency, Threads

- Synchronous execution dispatches from a single thread
- Parallel execution via threads or rate groups
- Sharing data and state requires locking and/or queues
- “Ships passing in the night”

- Implications:
  - Must plan concurrency model
  - Shared resources must be handled appropriately
  - Messaging and scheduling must be thought through
  - Care must be taken with work requiring specific timing

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Faults, FATALs, and Error Handling

- Flight software is expected to protect the spacecraft
- Off-nominal conditions will always occur; the universe desires this
- Spacecraft operators need to understand cause of behavior

- Implications:
  - Uncontrolled reboots and crashes should be avoided
  - Logging of off-nominal conditions should occur
  - Spacecraft should be made safe before loss-of-control resposes

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Modeling Flight Software in F´

Components

- Represent concrete function in the system
- Come in three variants: Active, Passive, and Queued
- Communicates with other components via ports

```
<component name="ActiveLogger" kind="active"
namespace="Svc">
  <import_port_type>…</import_port_type>
  <import_dictionary>….</import_dictionary>
  <comment>A component for storing telemetry</comment>
    <ports>
      <port name="LogRecv" data_type="Fw::Log"
kind="sync_input" >
        <comment>
        Telemetry input port
        </comment>
      </port>
      <port name="PktSend" data_type="Fw::Com" kind="output" >
         <comment>
          Packet send port
          </comment>
      </port>
       …
  </ports>
</component>
```

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Ports

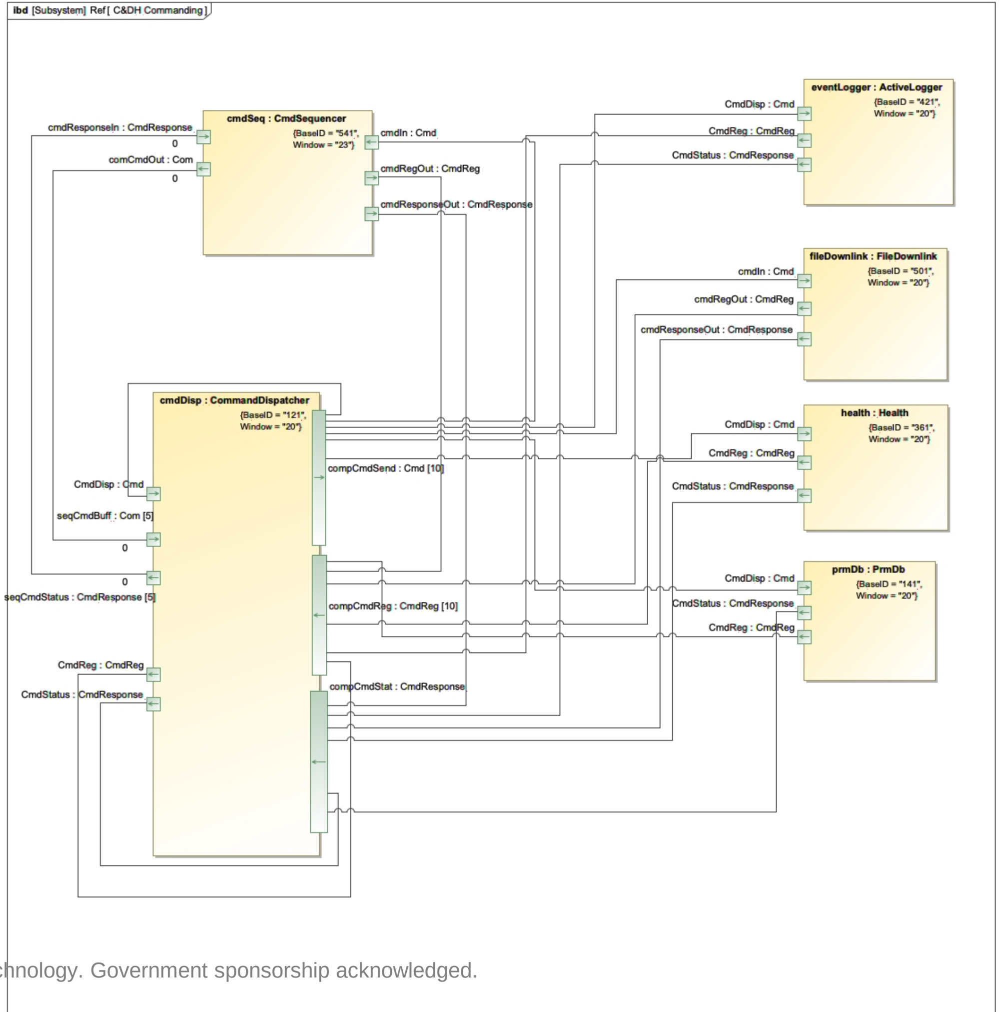
- Represent interface to component
- Form a point-to-point network for communication
- Have arguments with specific types

```xml
<interface name="Ping" namespace="Svc">
  <comment>
  Port for pinging active components
  </comment>
  <args>
    <arg name="key" type="U32">
      <comment>Value to return to
                  pinger</comment>
    </arg>
  </args>
</interface>
```

© 2019 California Institute of Technology. Government sponsorship acknowledged.

Topologies

- Topologies represent a network of components
- Contain instantiations of each component
- List connections between the ports of all the components



© 2019 California Institute of Technology. Government sponsorship acknowledged.

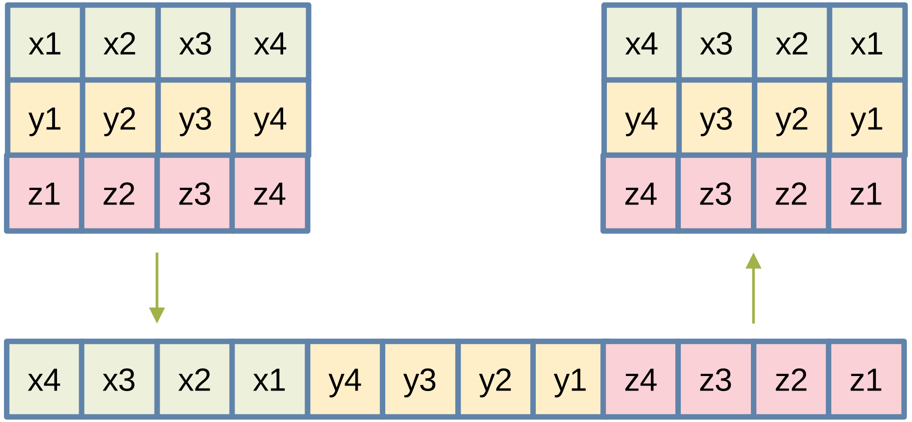
Data Serialization and Deserialization

- Data in RAM may be padded, expanded, or mixed with temporary values

- Bytes in RAM may have different orders across machines/devices

- Data in transit should be an array of bytes in specified order

```
struct Position3D {
    U32 x;
    U32 y;
    U32 z;
};
```



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Built-In F´ Functionality

Fw: Core Types

- Framework (Fw) provides the most basic F´ types and concepts
- Sized types U8, U32, F32, F64
- Buffers, Files, Logs, Command, other core data structures
- Assertions and other core handlers

- Fw
  - Buffer
  - Cfg
  - Cmd
  - Com
  - ComFile
  - Comp
  - FilePacket
  - Log
  - Logger
  - Obj
  - Port
  - Prm
  - Python
  - SerializableFile
  - Test
  - Time
  - Tlm
  - Types

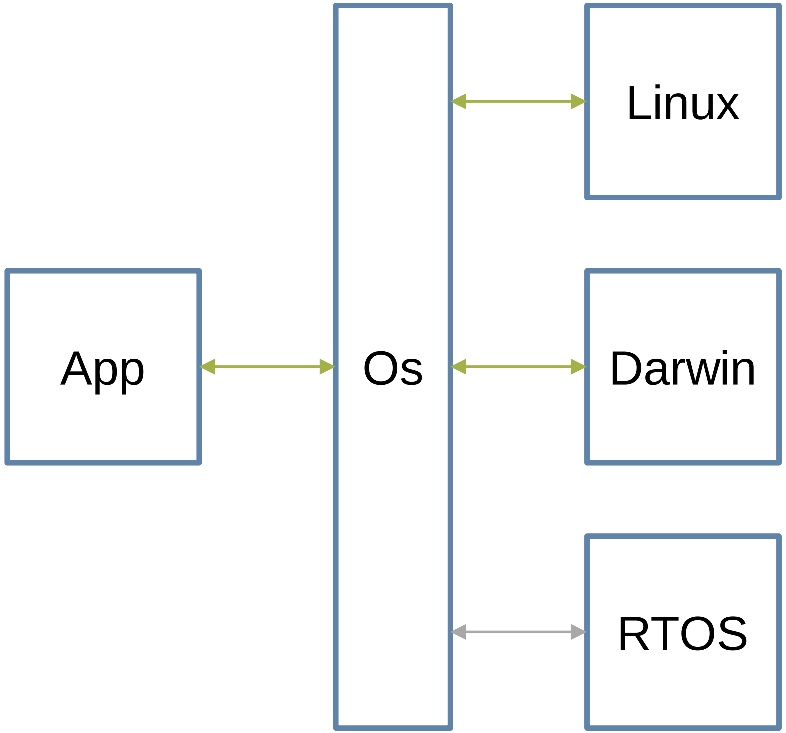
© 2019 California Institute of Technology. Government sponsorship acknowledged.

Os: Operating System Abstraction (OSAL)

- Operating call abstractions for consistent cross-compilation

- Files,File Systems, Queues, Tasks, Mutexes, and Timers

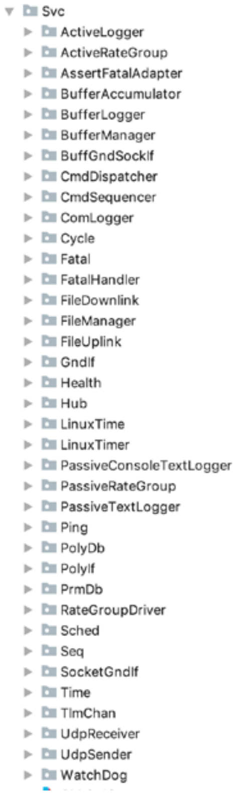
- Needs implementation for each Os type



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Svc: High-level service components

- Standard systems components enabling reuse
- F´'s "Standard Library"
- Events, Telemetry, Commanding, File Managements, Rate Groups

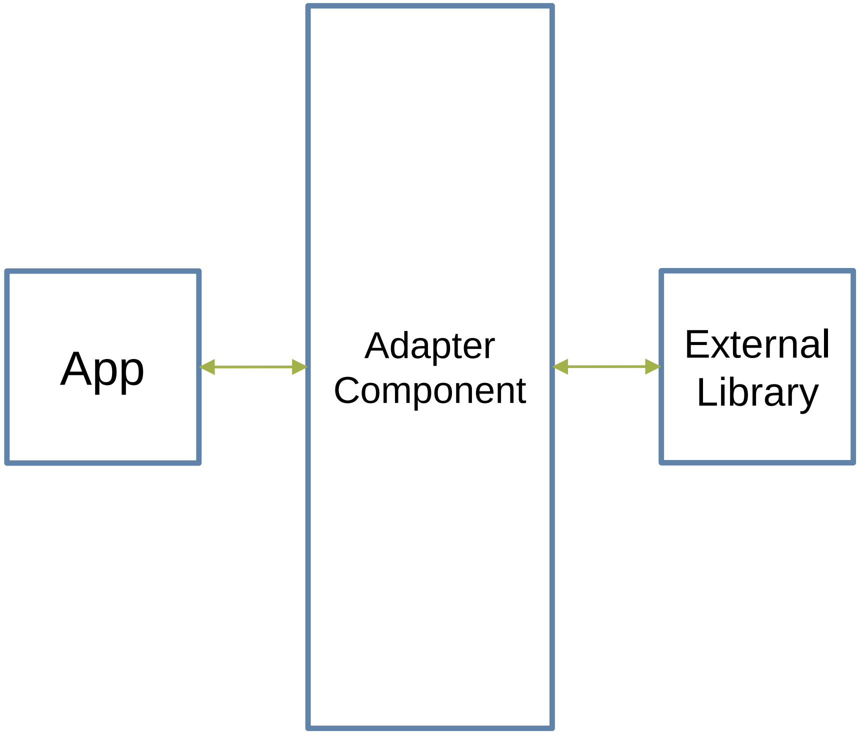


© 2019 California Institute of Technology. Government sponsorship acknowledged.

F´ Standard Patterns

Adapter Pattern

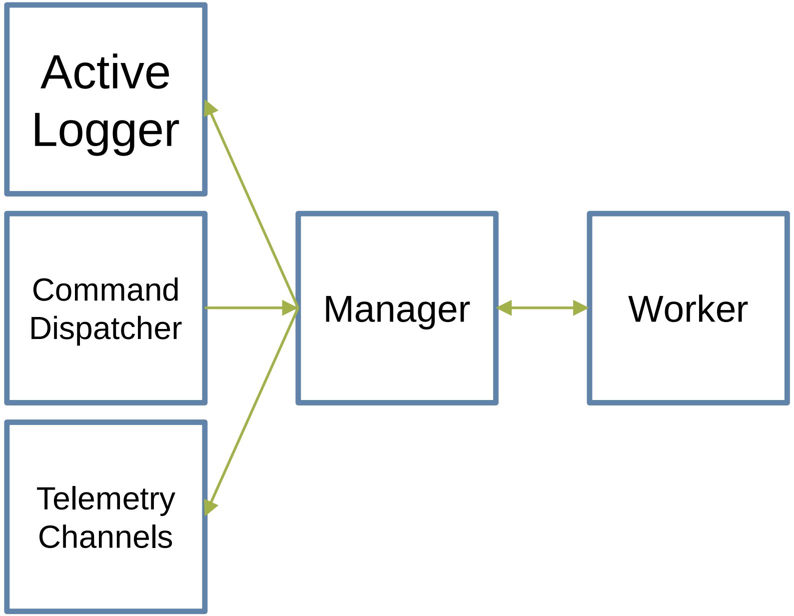
- Adapts “something else” to work with F´
- Typically done by writing an F´ component that bridges functionality, concurrency, timeliness, commands, events, and telemetry



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Manager - Worker

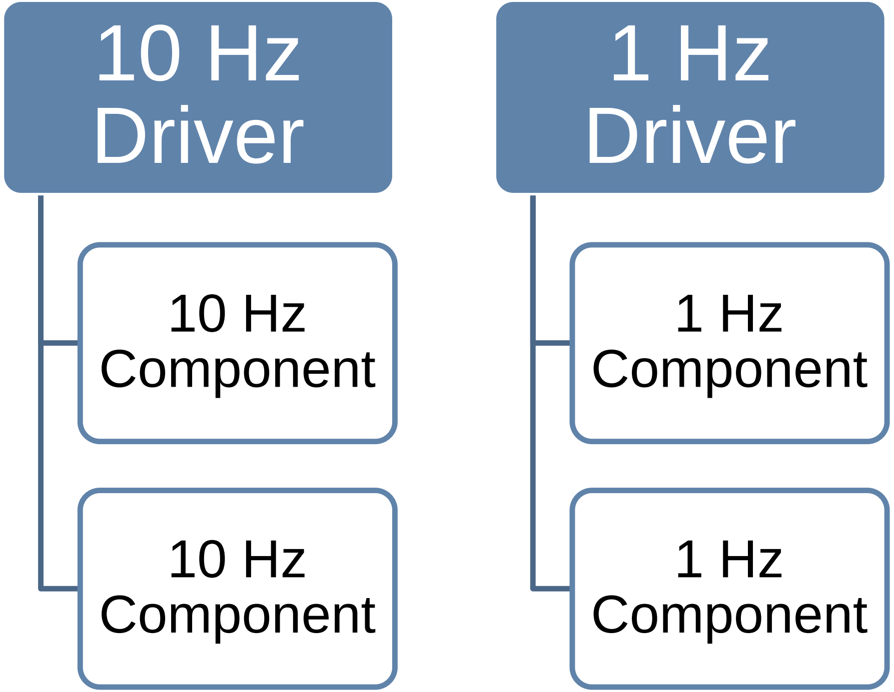
- Decouples long-running tasks from need for quick interaction
- Manager sends work to worker worker responds back afterwards
- **Only** Manager communicates with worker
- Parallels “worker thread” pattern



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Rate-groups Timeliness

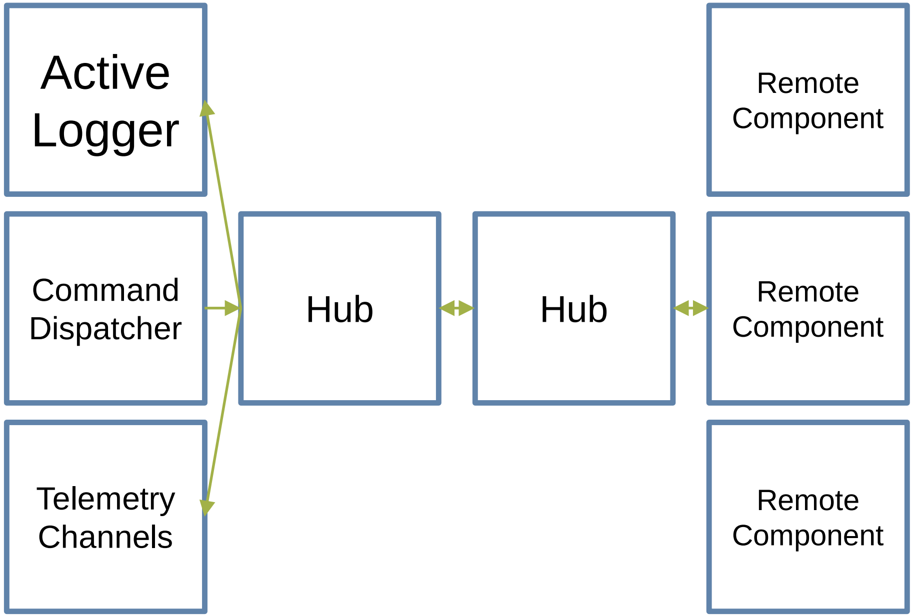
- Drives components at a set rate
- Simple provider of timeliness, allowing work at set time
- Care must be taken with other forms of concurrency
- Care must be taken to not slip



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Hub Pattern

- Routes multiple F´ ports across some communication layer
- Unwraps on the far side of the communication layer
- Allows for inter-process communication



© 2019 California Institute of Technology. Government sponsorship acknowledged.

Questions?

Lab Exercise

Jet Propulsion Laboratory
California Institute of Technology

jpl.nasa.gov