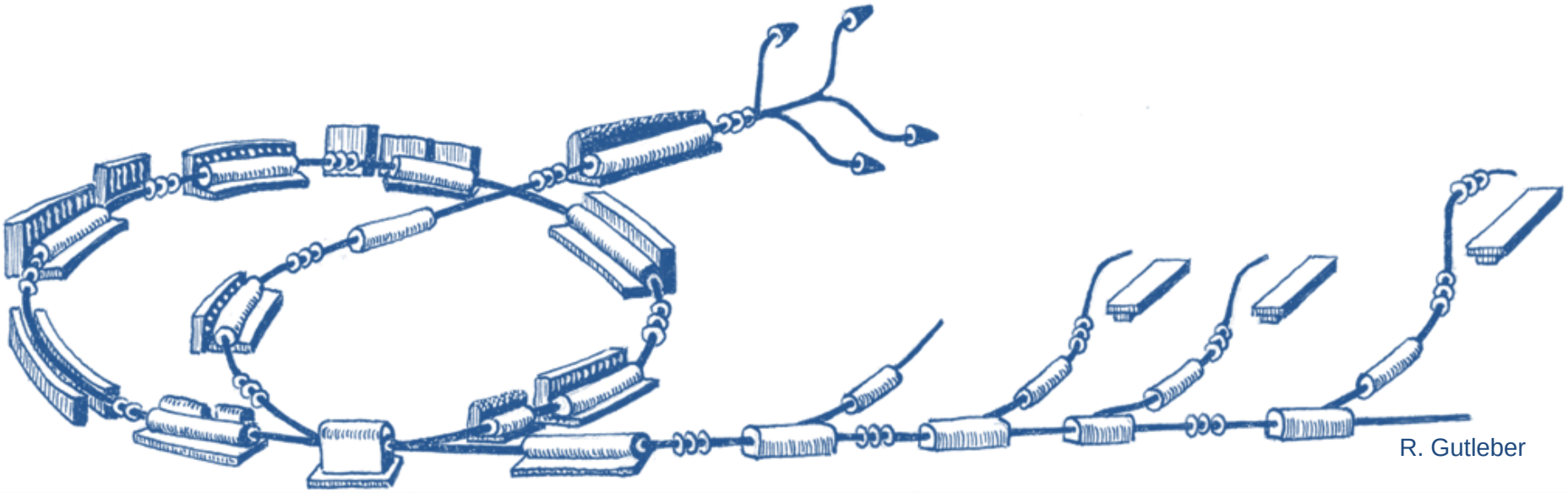
The Virtual Accelerator Allocator



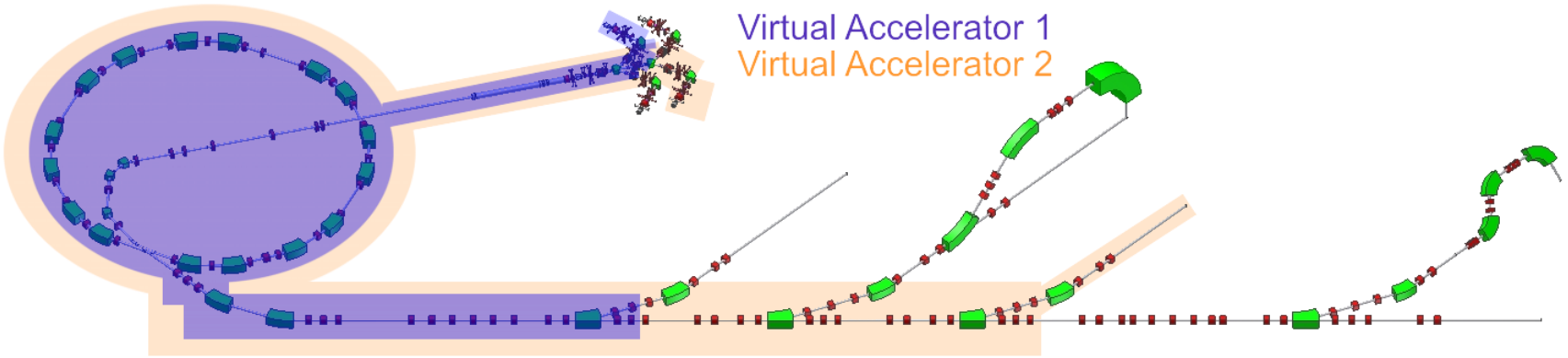
R. Gutleber

Overview

- Concepts
  - Virtual accelerators
- Architecture
  - Overview
  - BDCS-VAA comunication
  - VAA software
- Vaa Internals
  - Overview
  - Libraries Used
- Status

CONCEPTS

Virtual Accelerators (VAcc)



- Consists of a **list** of **machines**
- addressed with a unique identifier (e.g. CS-02-001-ACC)
- VAcc may be only in one mode
  - All Machines (Working Sets) have to be in that same mode
- Overlapping VAccs cannot be used at the same time.
- Non-overlapping VAccs can be used at the same time
  - And may be in different modes

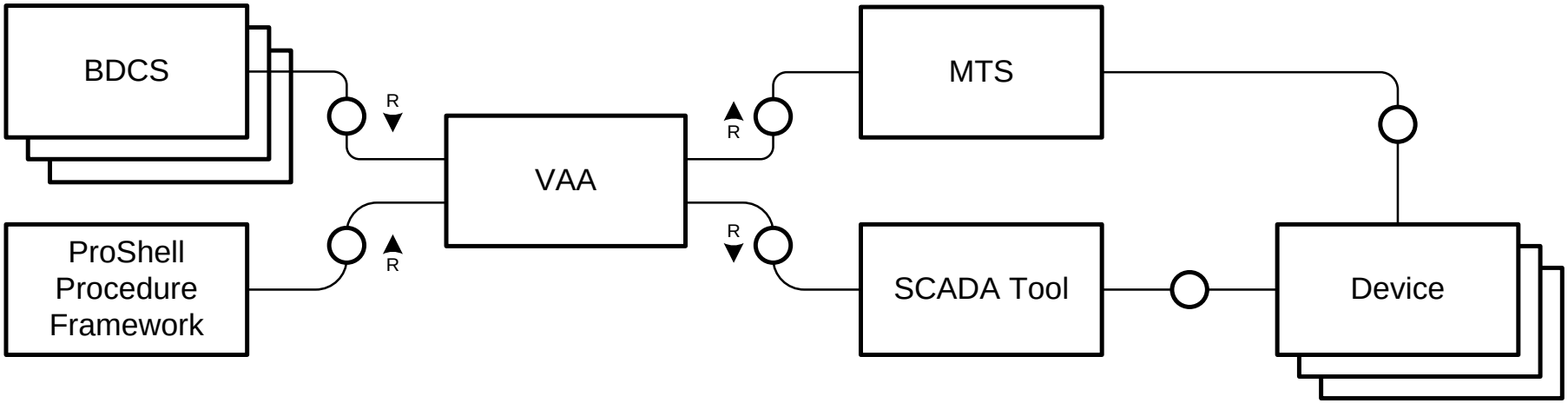
Virtual Accelerator Allocator (VAA)

- **Scheduler** that **allocates** VAccs for **exclusive** usage
- Scheduling based on **availability**, request time and priority
- **Receives** requests from clients (BDCS or ProShell)
- responsible for **configuring** all components of the VAcc
- **Forwards commands** (StartCycle, NextCycle, ...) from clients to the **MTS**

- VAA **will not automatically change** the **mode** (clinical, QA, physics, service) of any resource
  - Avoid unintendend mode changes
  - An **operator procedure** will be required to move a device in a certain state and mode

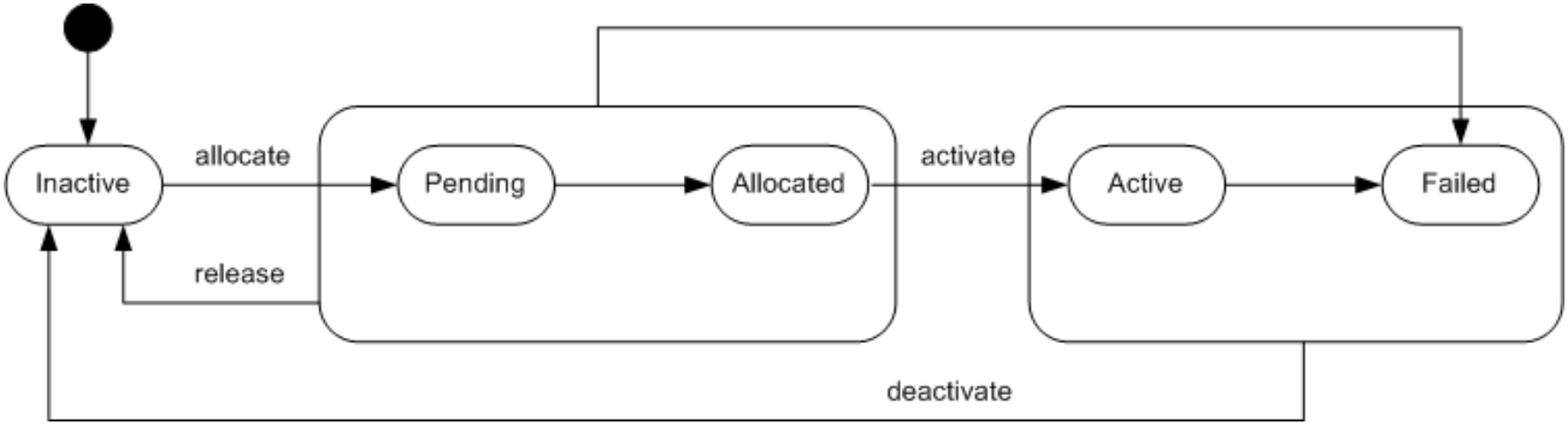
ARCHITECTURE

General Architecture



- VAA
  - **Forwards** requests to the **MTS** (NextCycle, StartCycle etc..)
  - **Single entry point** for allocation in the system (Safer, less error prone)
  - **Serializes requests** to MTS and PVSS (avoiding racing conditions)

VAA Request State Machine



Each allocation request is handled by a single VAA request state machine.

- **Pending** – resources may be allocated by another request
- **Allocated** – all resources are claimed
- **Active** – all resources are *configured* and *irradition* may start
- **Failed** – when one resource moved to failed
- **Inactive** – request not yet processed

Allocation

The VAA performs the following actions when receiving allocation request:

1. Check that none of the resources is in **failed state**
2. Check that all resources are in the **requested mode** (clinical, QA, physics, service)
   - Reject request if mode does not match
3. Check **availability** of all resources
   - **Keep** request **pending** if resources are in use by another request to **avoid starvation**.
4. **Allocate** all **resources**
5. **Configure MTS execution** slot (runfile, list of cycle-dependent devices)
6. Configure all resources to listen for MTS events in selected execution slot
7. Send **ActivateRun** command to MTS
   - To trigger loading of cycle-dependent values (waveforms)
8. Sends **„Prepare“ command** to all devices except beam transfer elements that connect to WSs outside of the VAcc (will unground and magnetize magnets etc..)

Note: Beam Transfer Elements are still disabled!

Activation

The VAA will perform the following actions on activation requests:

1. Check **state** (red,orange,green) of the **Patrol Control System** (PCS)
2. Sends **„Enable“ command** to all devices except beam transfer elements that connect to WSs outside of the Vacc.
   a. Move out **beam stoppers**
   b. Unground and magnetize all **switching dipoles**

Note: Subsequently the BDCS may request cycles for this VAcc

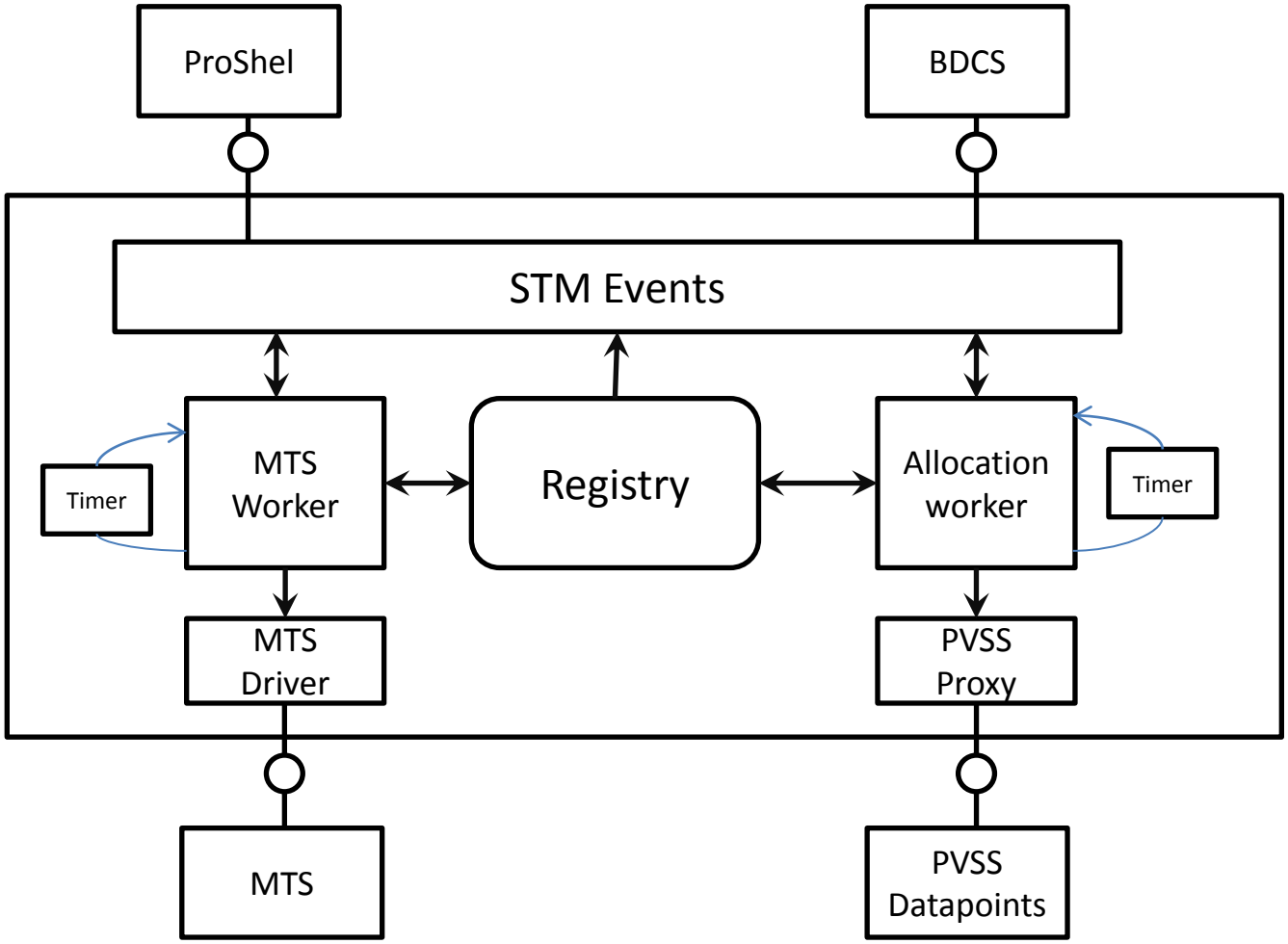
Deactivation

The VAA will perform the following actions on deactivation requests:

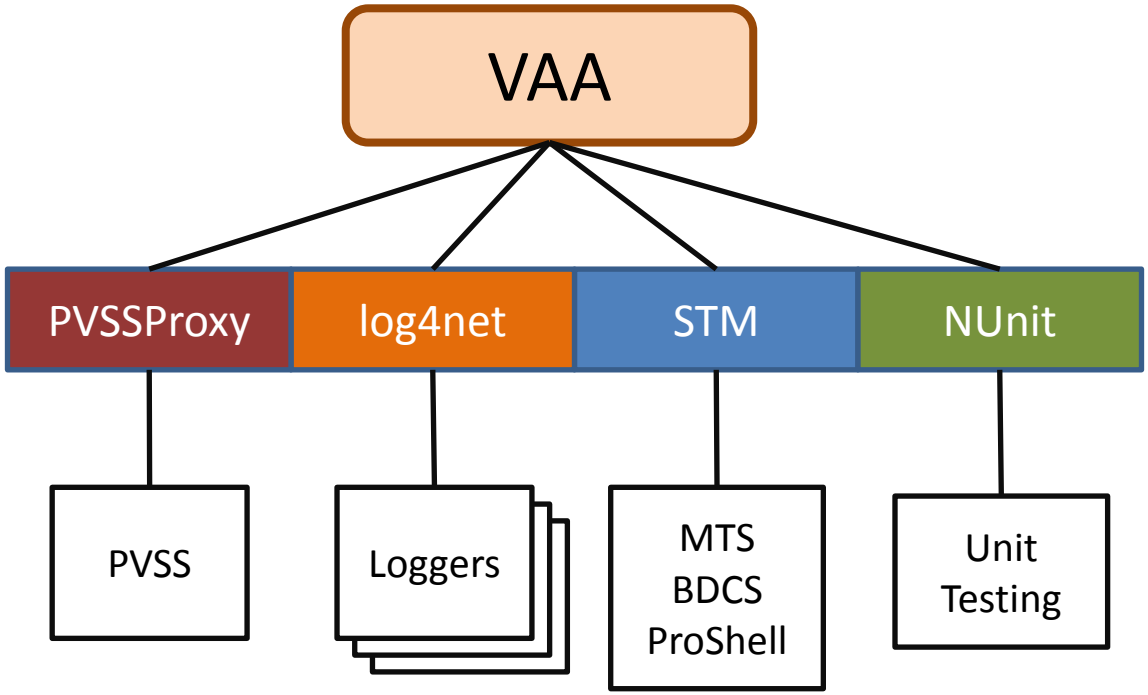
1. **Disable** all components in the VAcc
   a. Move in **beam stoppers**
   b. Ground and degauss all **switching dipoles**
2. **Free MTS** execution slot
3. Configure components to not listen to MTS events anymore
4. Send „Finalize“ command to all devices (degauss and ground magnets etc...)
5. **Deallocate** all components
6. Trigger **rescheduling** for pending requests

VAA internals

VAA Internal Diagram

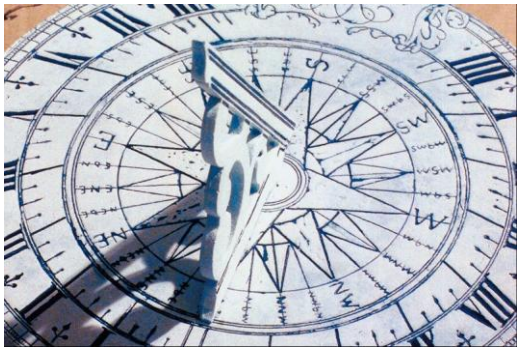


Libraries Used



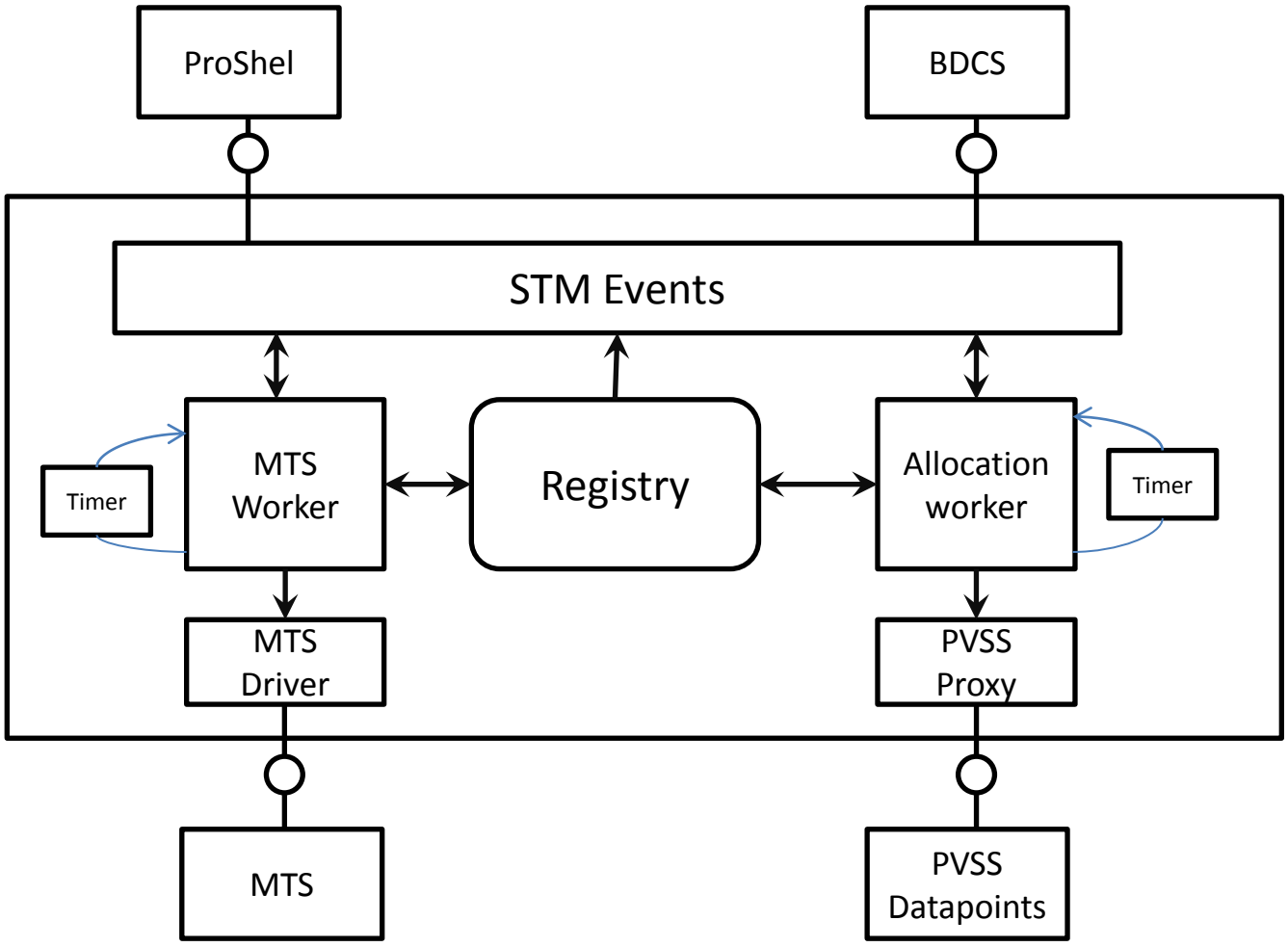
Development Enviroment

- C#, .Net framework 3.5
- Visual studio 2008
- Libraries
  - Simple messaging protocol STM (Cosylab)
  - PVSSProxy (MACS, see Angela's presentation)
  - Log4net
  - NUnit



Status And Outlook

VAA development Status



Current Status

- First draft of Enterprise Architect
  - Requirements gathering
  - Architecture design
- Prototype for Registry defined
- Work done in the allocation worker
  - Can get available Vaccs and WS from PVSS
  - Can recover state of allocation requests from PVSS (even pending and inactive requests)
  - Can claim WSs and VAccs (easy, does not perform the conflict checks etc...)
- Timer events
  - Only removes timed-out requests from the registry

Planed Work till next MACS week

- Requirements, Architecture and design finished
  - Ways of identifying which Vaccs are incompatible need to be refined
  - Ways of identifying Beam stoppers, Beam transfer elements in WSs has to be defined.
- VACC will follow the standard device state machine and react to command datapoints.
- VAA skeleton
  - Without STM based MTS communication
  - Without STM based supervisory interface (from BDCS)
  - With PVSS connection
  - Handling of conflicts and allocation of the devices

Planed Work till next MACS week

- Ahead Schedule
  - PVSS Connection
- In Time
  - Architecture Design
- Behind schedule
  - Requirements
  - Implementation
    - STM-SIM library
    - Interface to MTS

Additional Slides

BDCS-VAA communication

BDCS sends Allocation request

- containing *mode,* VAcc *identifier and runfile*
- Wait until VAcc changes to state **Allocated**

BDCS sends Activation request

- Wait until VAcc changes to state **Activated**

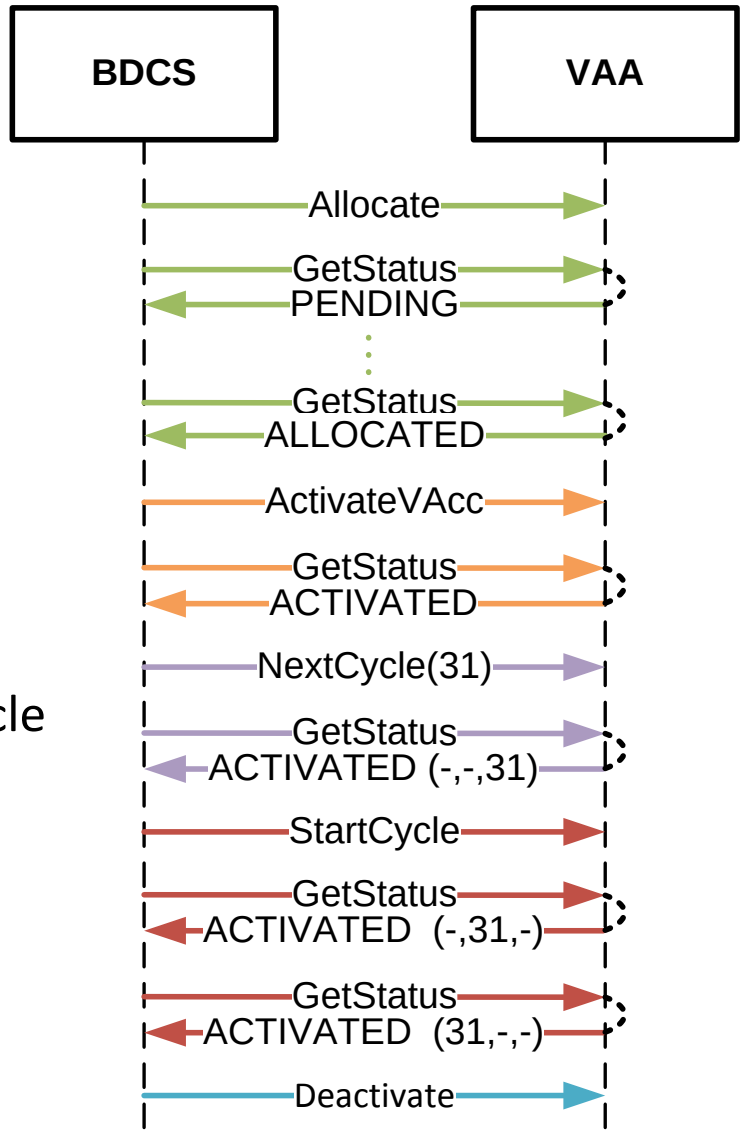
BDCS sends NextCycle command

- Wait until all devices configured for next cycle

BDCS sends StartCycle command

- Wait until cycle ended

BDCS sends Releases request



VAA-MTS Communication

VAA sends MTS commands that are distributed to all components through the Main Timing Receivers (MTR):

- ActivateRun
  - Runfile with a list of cycles
  - List of cycle-dependent components
  - Download cycle-dependent values (waveforms) from HTTP server to local harddisk and cache them into RAM.
  - Acknowledge when Runfile received by all components
- NextCycle
  - Trigger caching of next 2 cycles (e.g. Into the FPGA of the PCC)
  - Acknowledge when **first cyle is cached** for all components
- StartCycle
  - with cycle code
- DeactivateRun
  - **Clean** cycle cache in components