C2 Synchronization in Disconnected, Intermittent, and Limited (DIL) Environments

Primary: Topic 3: Data, Information, and Knowledge

20 June 2013

John McDonnell
John.mcdonnell@navy.mil
619-553-6098
5.3

Ryan Gabrys
Ryan.gabrys@navy.mil
619-553-6532
5.3

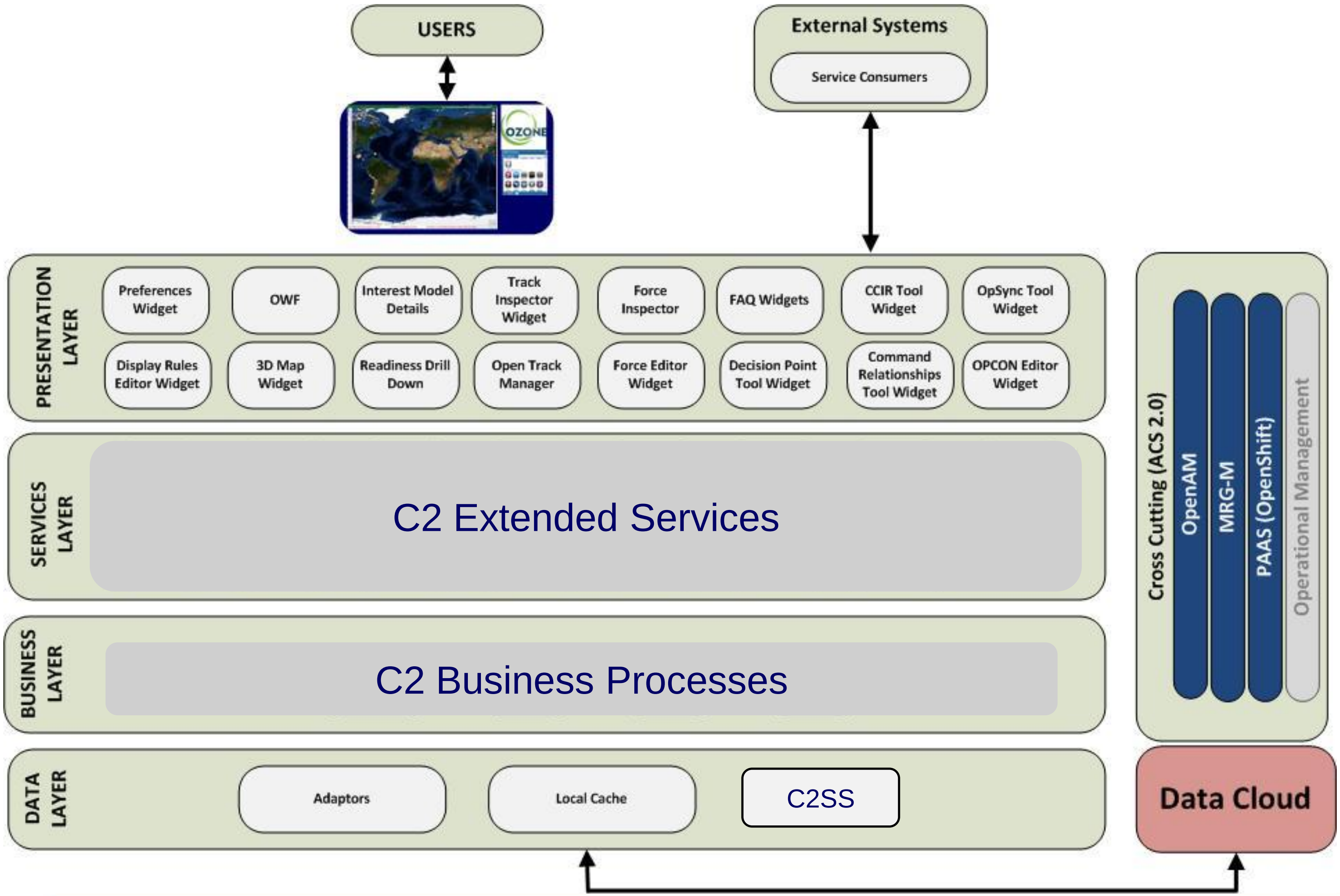
Project Objectives

- Summary of overall final goal of this work
  - Develop, integrate, and assess data synchronization techniques to support maritime tactical C2.
  - Provide a C2 Synchronization Service (C2SS) to support maritime C2
- Rationale for doing this work
  - Capability Gap:
    - Tactical C2 requires the capability to support collaborative planning, distributed execution and mission-focused data delivery in a DIL Environment
    - Data synchronization capabilities typically exist in Master-Slave and terrestrial environments.

Related Efforts

- FNT-09-04: Dynamic C2 for Tactical Forces and MOCs
  - DMS components provide prioritized synchronization of C2 data sources: (i) Rep/Sync Framework to support C2 data sources; (ii) Mission-based Prioritization and dynamic queuing; (iii) Responsive to observed network performance between relevant nodes
- Federation and Force Discovery Services (FFDS)
  - Set of services designed to facilitate discovery and sharing of information sources within a rapidly composable environment, while levying minimal development requirements on client applications and their developers.
- PMW150 SBIRs
  - Navy Wave: Collaboration based on Google Wave/Operational Xforms
  - True Numbers: Data Pedigree and Provenance

C2 Synchronization Service



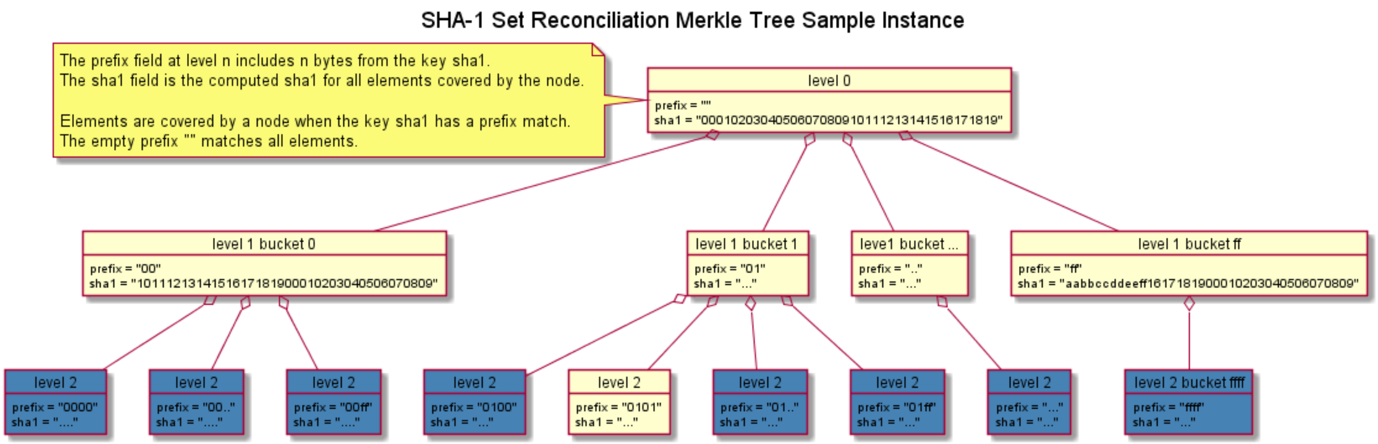
APPROACH

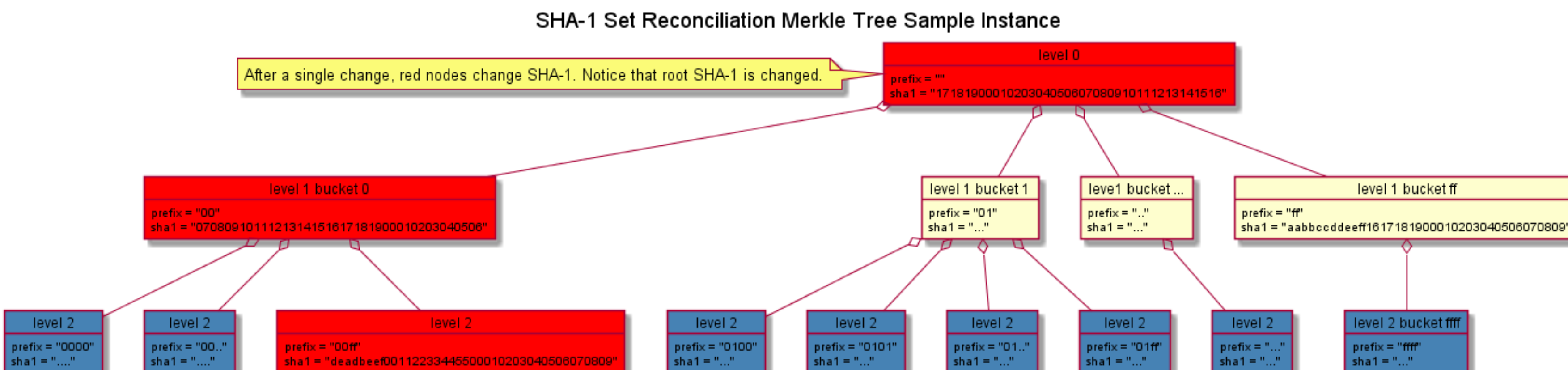
Develop, integrate, and assess data synchronization techniques to support maritime tactical C2-data consistency in a DIL environment

- Develop target architecture
  - Initial technology survey resulted in a taxonomy of 14 generic C2SS functions
  - Working taxonomy provides a basis for developing a C2SS functional architecture
- Develop/Implement technologies
  - Vector Clock
  - Hash Representations: SHA-1/Merkle Trees
- Focus On
  - Concurrent Distributed Ops
  - Conflict ID
  - Data Integrity
  - Eventual Consistency
- Support track management and planning/tasking applications
- Test and assess in lab tests and/or exercises (eg. Trident Warrior)

Merkle Trees w/ SHA-1 Encodings

SHA-1 is used to determine data consistency w/in a Hash Tree:

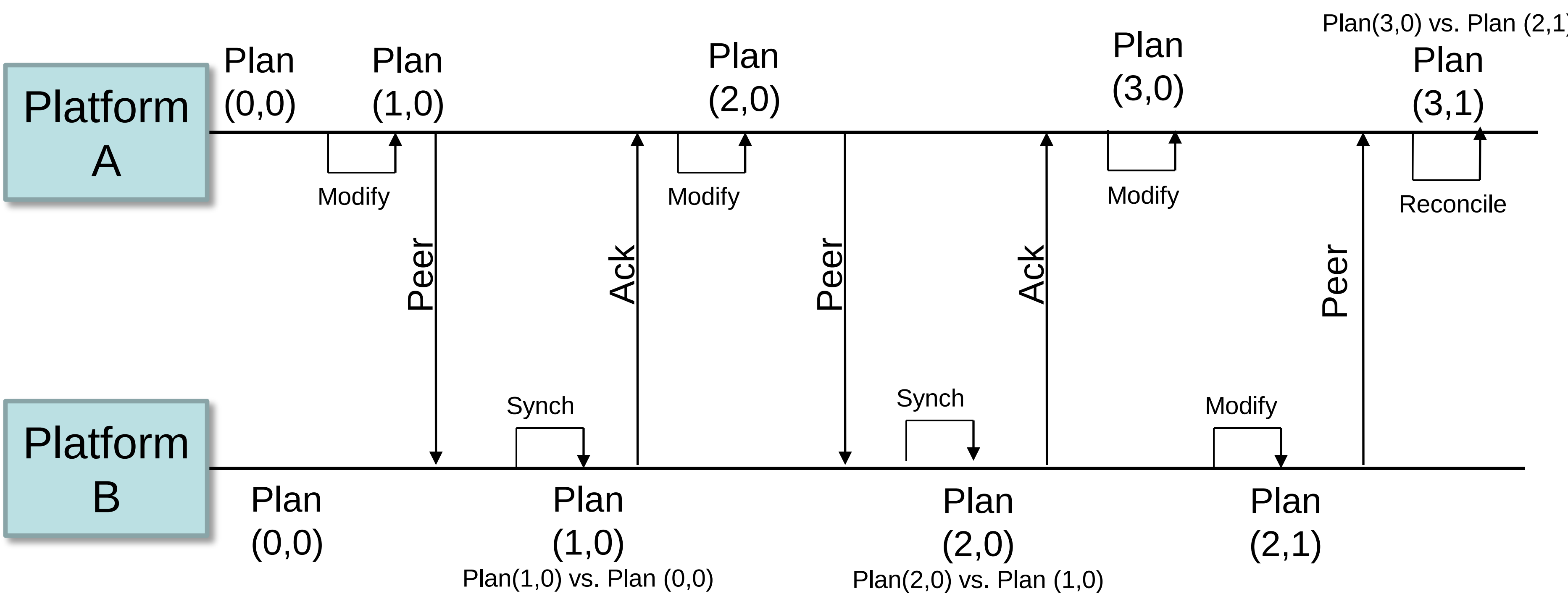




Vector Clocks

Vector clocks support the causal ordering of events and are described by the partial ordering property:

$$VC(x) < VC(y) \leftrightarrow \forall z[VC(x)_z \leq VC(y)_z] \wedge \exists z'[VC(x)_{z'} < VC(y)_{z'}]$$



Implementation of VC as Key/Version Pairs

▼ A vector of versions

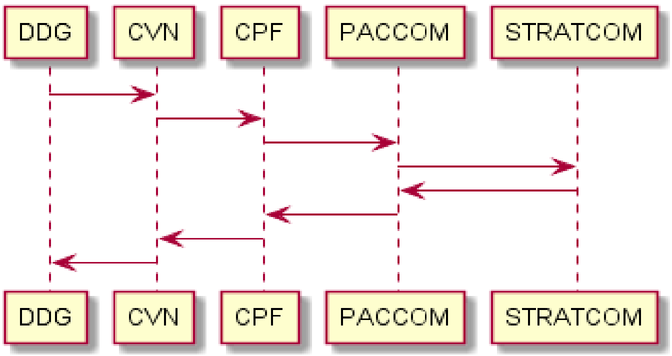
- Element at index *i* in the vector represents the version at node *i.*
- Updating element on node *i* increments the version at vector position *i*.
- Make sparse
  - Use map of ***node name*** to ***version***. If nodes are named DDG and CVN, then {DDG=117, CVN=4}.
  - This means DDG has made 117 changes and CVN has only made 4 changes. All other nodes are implied at version 0.

- **Examples**
  - {DDG=117,CVN=4} **=** {DDG=117,CVN=4}
  - {DDG=117,CVN=4} **<** {DDG=117,CVN=5}
  - {DDG=118,CVN=4} **conflicts with** {DDG=117,CVN=5}

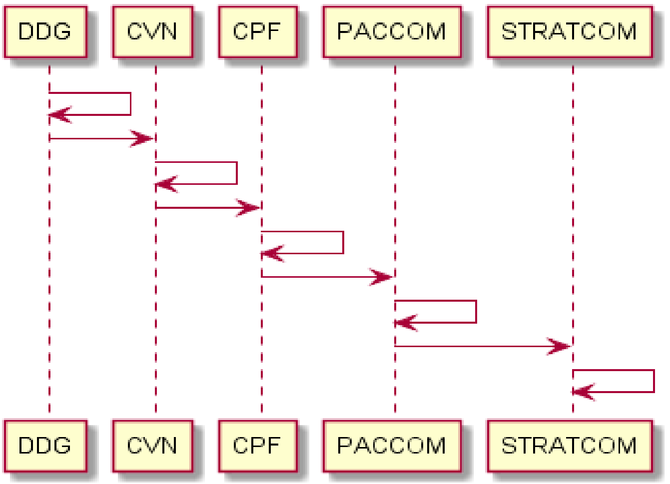
Multi-Master Updates

- Prior research regarding synchronization of track data used the notion of master-mirror to manage the direction changes flowed.
  - Differences are always an add, update, or delete at the mirror.
  - Changes are pushed to “Top COP”, then back
- DIL networks require that edits are applied at any node to support availability, then reconciled.
  - Any node can “master” a change (multi-master).
- Technical issues:
  - It is hard to determine last edit
  - No common time source, computer clocks drift.
- Vector Clocks indicate when there is a “causal ordering” of changes or if the changes are in “conflict”.

VDX Client Update

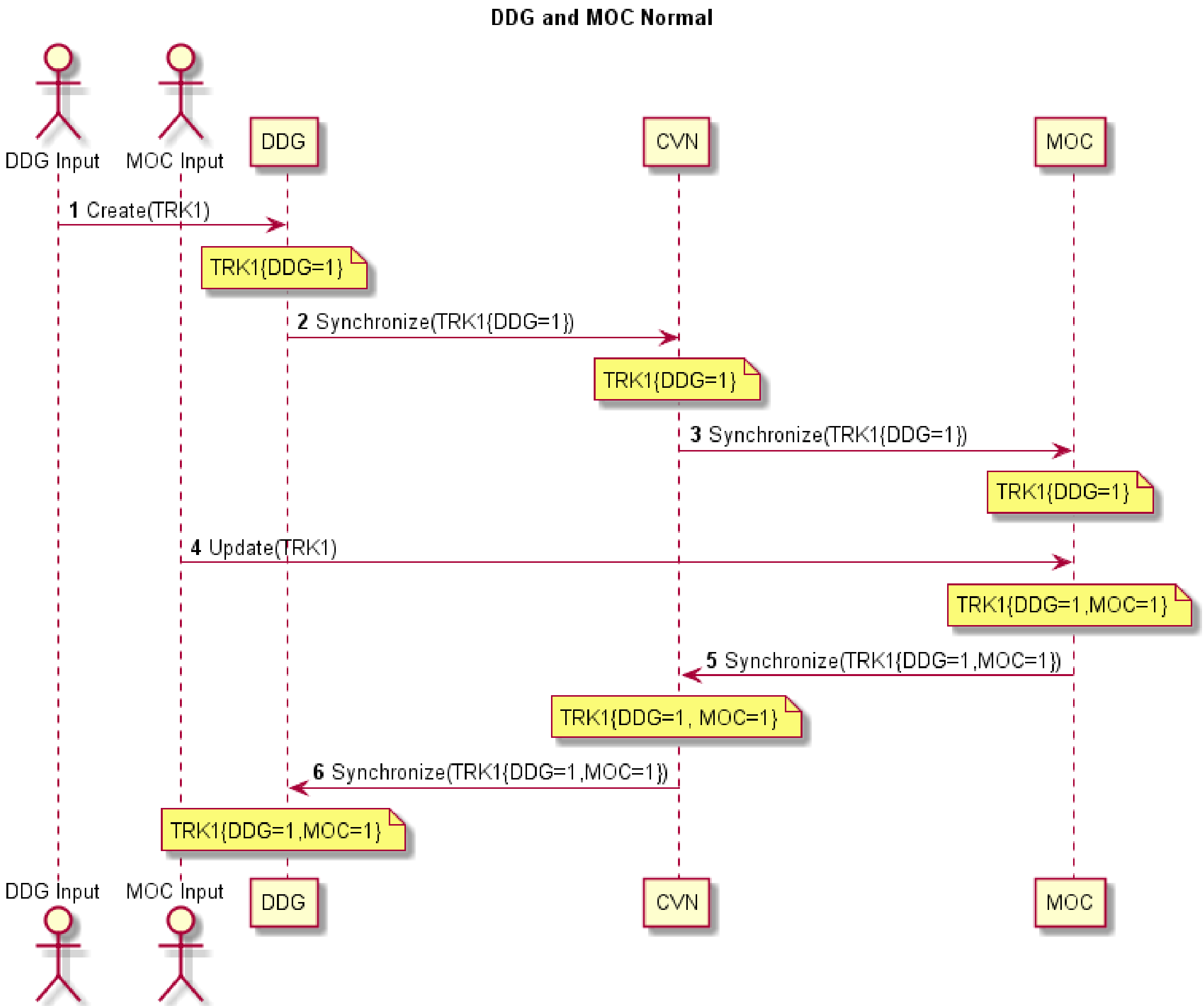


Multi-Master Client Update



Track Updates

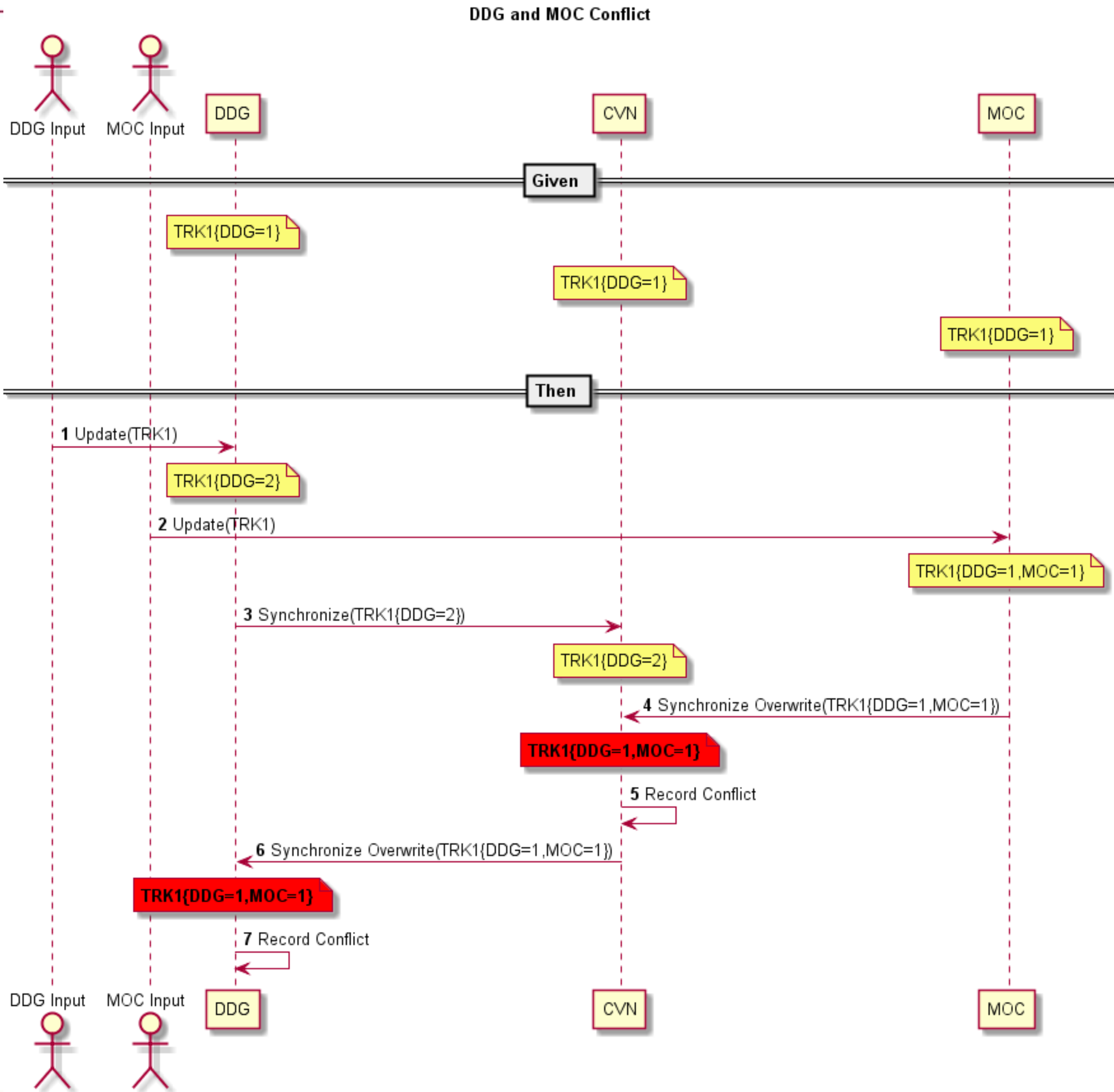
1) DDG creates TRK1 so Vector Clock (VC) becomes TRK1{DDG=1}
2) Synchronizing DDG/CVN sees TRK1 with empty VC at CVN, sends update
3) Synchronizing CVN/MOC sees TRK1 with empty VC at MOC, sends update
4) MOC updates TRK1 so VC becomes TRK1{DDG=1,MOC=1}
5) Synchronizing CVN/MOC sees TRK1{DDG=1} at CVN and TRK1{DDG=1,MOC=1} at MOC which dominates, so update CVN
6) Synchronizing DDG/CVN sees TRK1{DDG=1} at DDG and TRK1{DDG=1,MOC=1} at CVN which dominates, so update DDG



Track Updates w/ Conflicts

- Given DDG, CVN, and MOC all have TRK1{DDG=1}

1) DDG updates TRK1 so Vector Clock (VC) becomes TRK1{DDG=2} at DDG
2) At roughly the same time, the MOC updates TRK1 so VC becomes TRK{DDG=1,MOC=1} at MOC
3) Synchronizing DDG/CVN sees TRK1{DDG=2} dominates TRK1{DDG=1} so DDG sends TRK1{DDG=2} to CVN
4) Synchronizing CVN/MOC sees conflict TRK1{DDG=2} versus TRK1{DDG=1,MOC=1}, potentially resolve using higher echelon
5) Conflict recorded to present to User Interface
6) Synchronizing DDG/CVN sees conflict TRK1{DDG=2} versus TRK1{DDG=1,MOC=1}, resolve using higher echelon
7) Conflict recorded to present to User Interface

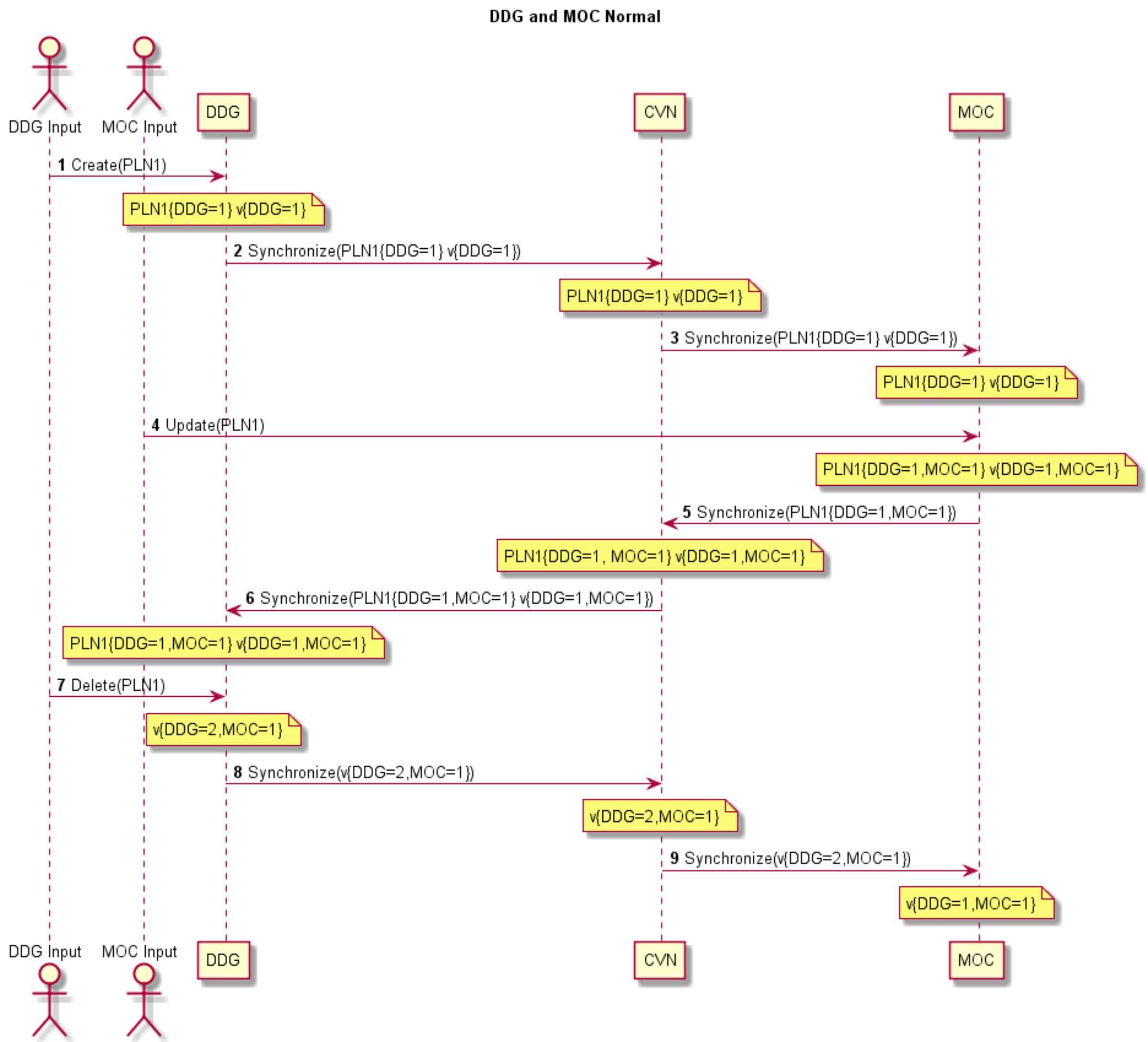


Tombstones and Vector Clocks

- OTM keeps Vector Clocks for deleted tracks
  - Used to determine delete or add action
  - Prunes Tombstones after one month
    - TRK9{DDG=12}, TRK2{DDG=3}, ~~TRK1{DDG=17,CVN=4}, TRK7{CVN=3}~~
- PTDS uses a modified scheme for Vector Clocks based on
  - Bieniusa, etc…, *An Optimized Conflict-free Replicated Set*, Octobre 2012
    - Single version number per node
    - Version vector, ***v***, with latest version seen from every node.
      - **Not a tombstone for every element**
    - Same example from above would look like
      - TRK9{DDG=12}, TRK2{DDG=13}, v{DDG=17, CVN=4}
      - **When many deletes much less storage**

PTDS Add/Update/Delete using ORSET

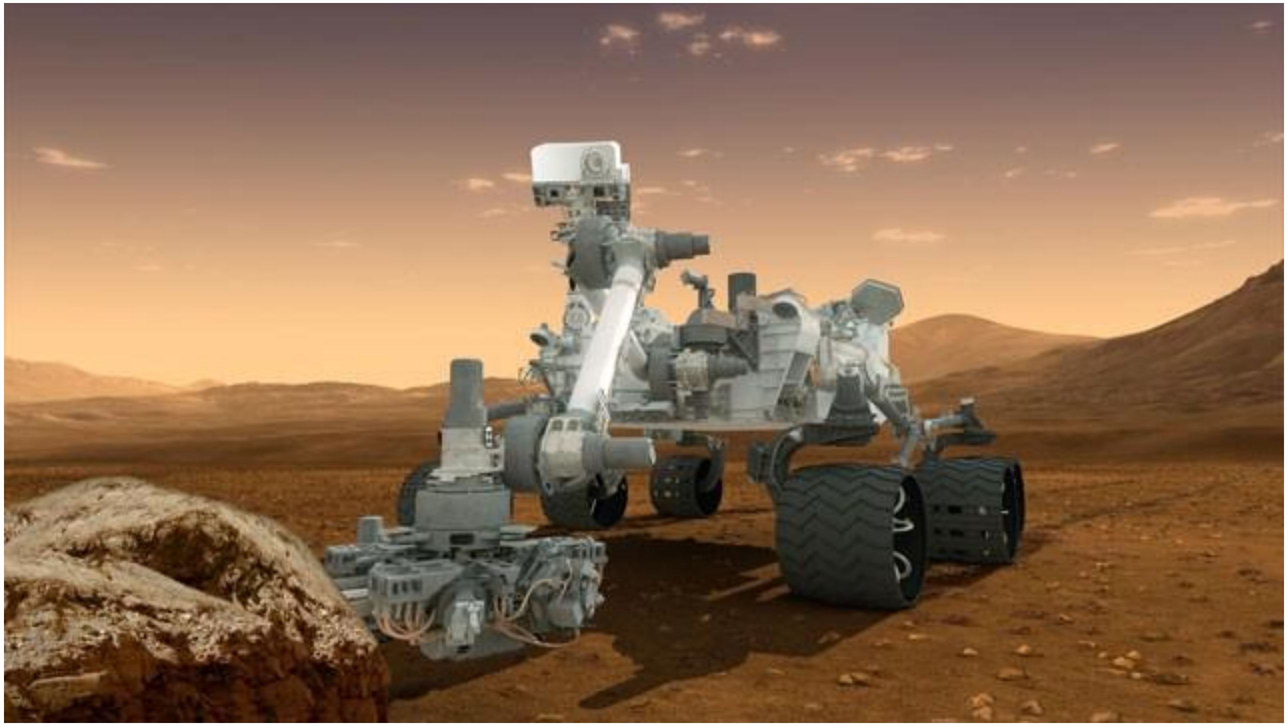
1) DDG creates PLN1 so ORSET becomes PLN1{DDG=1} v{DDG=1}
2) Synchronizing DDG/CVN sees empty ORSET at CVN, sends update
3) Synchronizing CVN/MOC sees empty ORSET at MOC, sends update
4) MOC updates PLN1 so ORSET becomes PLN1{DDG=1,MOC=1} v{DDG=1, MOC=1}
5) Synchronizing CVN/MOC sees TRK1{DDG=1} at CVN and TRK1{DDG=1,MOC=1} at MOC which dominates, so update CVN
6) Synchronizing DDG/CVN sees TRK1{DDG=1} at DDG and TRK1{DDG=1,MOC=1} at CVN which dominates, so update DDG
7) DDG deletes PLN1 => ORSET empty values, vector v holds {DDG=2,MOC=1}
8) Synchronizing DDG/CVN sees empty values and v dominates, so delete
9) Synchronizing CVN/MOC sees empty values and v dominates, so delete



FY13 Work Remaining

- PTDS
  - Testing using OpSync Tool UI
  - Web configuration
  - DMS side by side
- OTM
  - TW13 Participation
  - Conflict Resolution: Heuristics (Multi-Echelon rules)
- Analyze Performance
  - Topology trade-offs (Hierarchical Master/Mirror, P2P, Hybrid)
  - Granularity trade-offs to reduce conflicts/bandwidth requirements
- Develop MOEs/MOPs
- Testing
  - A DMS/C2SS shared Test Bed is being developed at SSCPAC to support UNCLASS C2RPC including OTM and PTDS applications.

Questions?



Rep/Synch Models

- Replication is the process of keeping a set of replicas consistent as they evolve over time.
- Reconciliation is the process of computing the symmetric difference between two sets.
- Synchronization
  - Master-Slave
  - Peer-to-Peer
  - Multi-Master

CAP Theorem

- When designing distributed web services, there are three properties that are commonly desired:
  - Consistency
  - Availability
  - Partition Tolerance
- It is thought that it is impossible to achieve all three, but is has been proven that eventual consistency can be achieved. [1]
- The COP has adopted a model of satisfying availability and partition tolerance while sacrificing consistency. The COP sacrifices consistency during network partitioning so that the local command can operate from its local sensors.