Distributed Interactive Simulation (DIS) 101: The Basics

Don Brutzman and Don McGregor

Modeling, Virtual Environments, Simulation (MOVES) Institute

Naval Postgraduate School (NPS)

Monterey California USA

brutzman@nps.edu

The learner will be able to…

➢ Identify what standards are used by distributed simulations for military use.

➢ Identify what types of communication protocols are used for various networks.

➢ Identify what aspects needs to be standardized, and what aspects can be customized, to support diverse simulations with differing models and goals.

➢ Identify how DIS techniques for dead reckoning (DR), visual smoothing and distributed collision detection can reduce network traffic.

- ➢ What is distributed simulation?
- ➢ Military modeling & simulation distributed simulation standards
- ➢ Underlying TCP/IP networks
- ➢ DIS: Entity State PDUs
- ➢ DIS: Entity Types and Entity IDs
- ➢ DIS: Coordinate Systems
- ➢ DIS: Shooting, Dead Reckoning, Smoothing, Visual Synchronization
- ➢ DIS and Open-DIS: DIS version 8 Development, Implementation Efforts
- ➢ Bibliography

➢ A *distributed simulation* runs on multiple cooperating hosts on the network

➢ *State information* describes the position, orientation, and other information about an entity at a point in time

➢ *Live, Virtual, Constructive* (LVC) simulation involves different hosts doing different aspects of simulation in one cooperative system

- Live: Real people, real systems
- Virtual: Real people, simulated systems (human in loop)
- Constructive: simulated people, simulated systems (AI controlled)

➢ Automatic Identification System (AIS) is a standard for transmitting the current position of commercial ships in the real world; ships have a transmitter and receiver on board and send information in a standard format. This is a *live* component (real system, real people)

➢ Ship simulators portray a simulated virtual view of navigation from the perspective of a ship's bridge, perhaps in a "cave" environment with wall-sized screens. This is a *virtual* component (real people, simulated system)

➢ We can also inject simulated, computer-generated ships controlled by AI into the simulation. This is a *constructive* component (simulated system controlled by simulated people)

Virtual: bridge simulator

Live: AIS feed

Single
Shared LVC
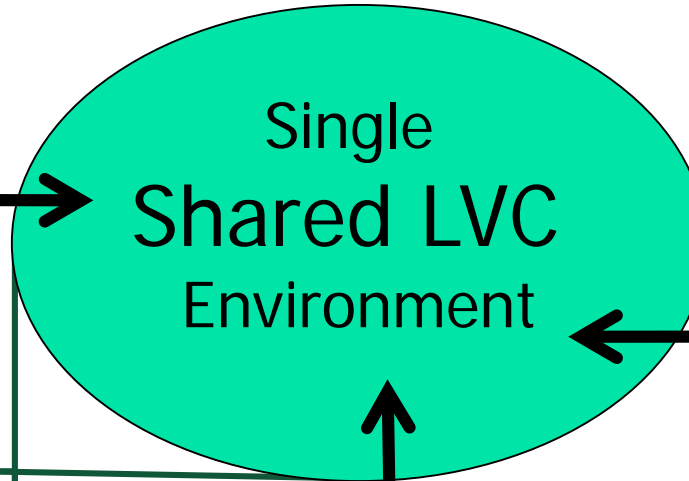Environment

Participants may be
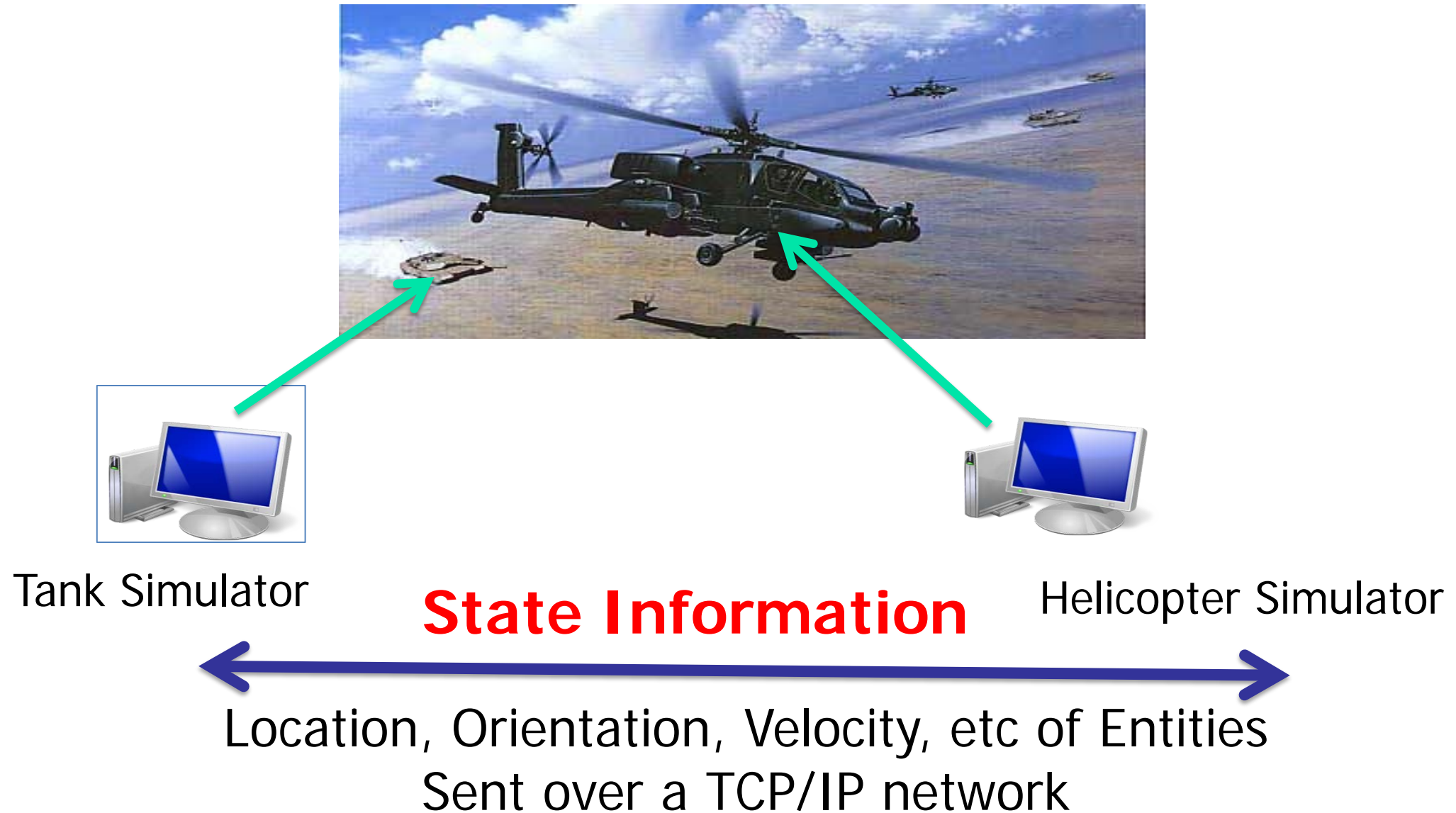local or distributed

Constructive: computer-
generated ship traffic,
controlled by AI agents

What Do We Want to Do?

- ➢ People who want to learn about Distributed Interactive Simulation (DIS) usually are in the "virtual" or "constructive" domains, though it can also be used in the live domain

- ➢ They want to simulate ships, tanks or other entities in the 3D world, controlled by humans or AI

- ➢ We need to exchange *state information* between hosts on the network about entities in the world
  - ▪ To view someone else in the simulation, we need to know their position, orientation, and other state information, and this information needs to be sent to us by them

- ➢ Typically entities are controlled by different hosts that are connected via a network. The state information is usually exchanged over the TCP/IP protocol

Tank Simulator

State Information

Helicopter Simulator

Location, Orientation, Velocity, etc of Entities
Sent over a TCP/IP network

- ➢ The format of state update messages needs to be *exactly* specified.
  - ■ What coordinate system should you use? You need one that works well for ground and air systems, and can handle curvature of the earth issues.
  - ■ Text or binary format messages?
  - ■ The order in which the fields appear?
- ➢ Network issues: what happens if a message is dropped?
- ➢ Scalability issues: how can we get a reasonable number of entities to participate?
- ➢ Latency: can we keep the message delay reasonable?
- ➢ Do all participants have a consistent, coherent operational picture?

➢ There are many ways to exchange the state information, but we want a *standard* way so we can interoperate with simulations from many vendors rather than being locked in to one. We don't want to use only one vendor's proprietary method, for which we will pay dearly

➢ In the case of defense modeling and simulation, the big three are

- TENA: Test and Training Enabling Architecture

- HLA: High Level Architecture

- DIS: Distributed Interactive Simulation

➢ TENA and HLA borrow many semantic concepts from DIS. Understanding DIS has many carry-over benefits when working with other standards.

The controlling host sends the state information update via one of the standards to other hosts

Controlling Host

| DIS |
| HLA |
| TENA |

Receiving Host
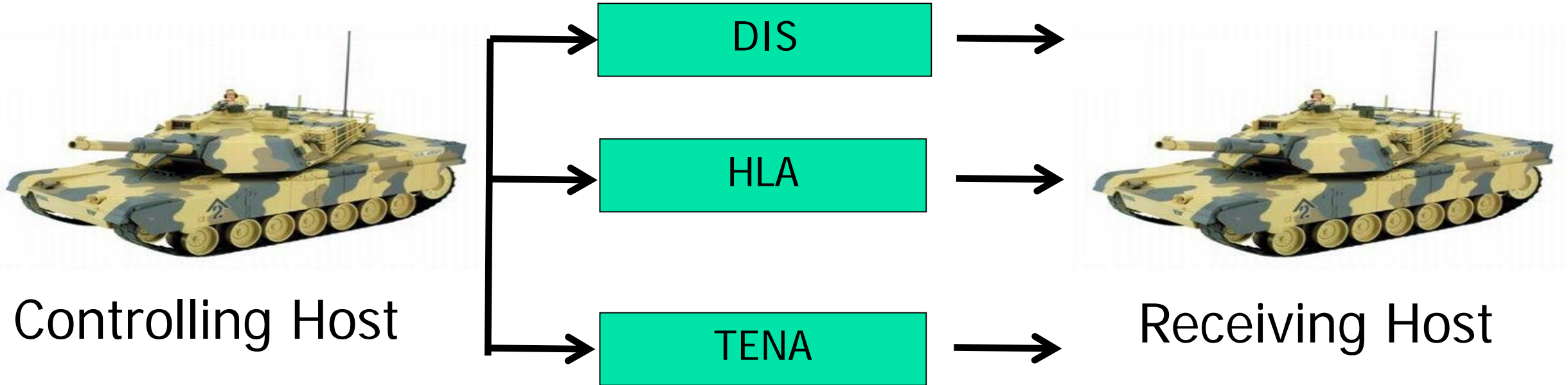
State Information Updates

- ➤ Used on ranges; often the "L" in Live-Virtual-Constructive simulations
- ➤ Designed for real time and embedded systems, real sensor systems, etc.
- ➤ In effect it is thinly disguised CORBA distributed objects with multiple new features to help it work in a simulation environment
- ➤ Can gateway it to other standards, such as DIS or HLA
- ➤ See http://tena-sda.org

- ➢ HLA is very general and intended to cover most defense modeling domains including training, analysis, and engineering in addition to virtual worlds

- ➢ Participants communicate via an agreed-upon *Federation Object Model* (FOM) and an API associated with a *Run-Time Infrastructure* (RTI)

- ➢ Specification is maintained by SISO (http://sisostds.org), is IEEE standard 1516 and has implementations by

  - MAK (http://www.mak.com),
  - Pitch (http://www.pitch.se),
  - Portico (older version) (http://porticoproject.org) and others

- ➢ http://www.pitch.se/hlatutorial is a good introduction to HLA
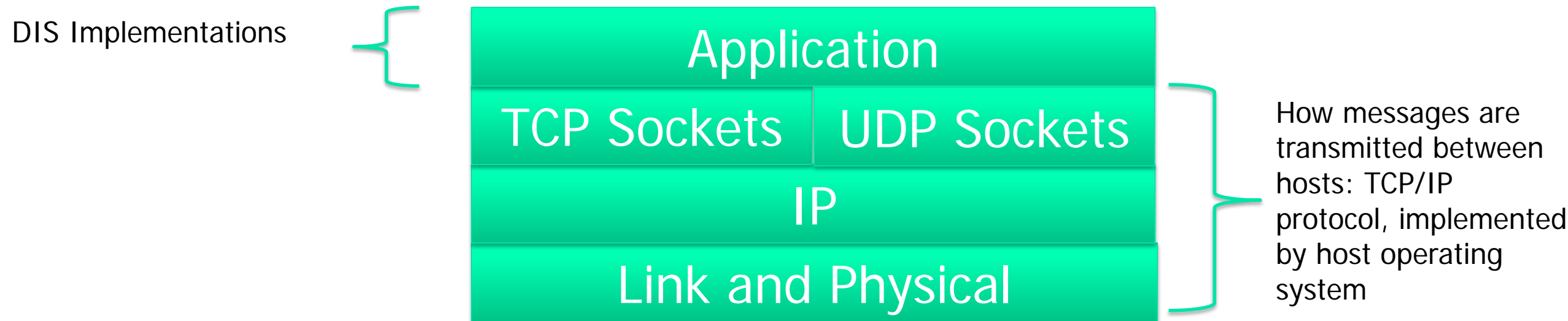
- ➢ DIS was the first standard to tackle these problems in a systematic way

- ➢ Originated in the SIMNET project in the 80's. DARPA supported converting the SIMNET research into a standard; SISO developed the standard and took it to IEEE for approval.

- ➢ This means anyone can get the standard from IEEE, implement it, and participate in a simulation

- ➢ Development of standard continues; updated DIS standard version 7 is the latest approved version. SISO maintains the standard and presents it to IEEE for standards approval

- ➢ Substantial commercial support, open source implementations, many home grown implementations of portions of the standard

➢ What does DIS support?

- A standardized way to exchange messages about entities in a virtual world

- Common semantics for coordinate systems and other information, such as how to describe and specify entities

- Common practices to ensure interoperability

> ➤ DIS defines the format of the messages, but doesn't specify how to get the messages from one host to another. Almost always this is done via TCP/IP

DIS Implementations

Application
TCP Sockets	UDP Sockets
IP
Link and Physical

How messages are transmitted between hosts: TCP/IP protocol, implemented by host operating system

➢ TCP sockets have higher latency, higher *jitter* (variation in latency), and doesn't scale to large numbers of hosts as well

➢ UDP sockets have lower latency, lower jitter, scales to large numbers of participants better, but are unreliable

➢ DIS typically uses UDP

➢ Hold on—UDP is *unreliable*? What's up with that?

- Individual UDP messages may be dropped by the network; there's no guarantee that a UDP message will be delivered. This is a tradeoff to achieve lower latency and jitter, and better scalability
- This is not as big a deal as it might seem. If we're getting position updates from an entity every 1/30th of a second, does it matter if we drop one? We have better information coming along shortly, so why try to resend the dropped message?

- We know what we want to do: send a message that tells another host what the state of an entity is

- We know how we do this: we send the message via TCP/IP, typically in a UDP message over broadcast or multicast

- What is the actual format of the messages? This is specified by the DIS standard.
  - There are dozens of possible messages to send, relating to everything from logistics to electronic warfare to radio communications
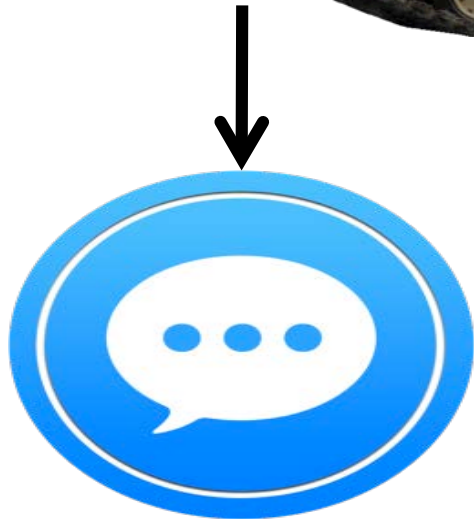 - Each message type is called a "Protocol Data Unit" (PDU)
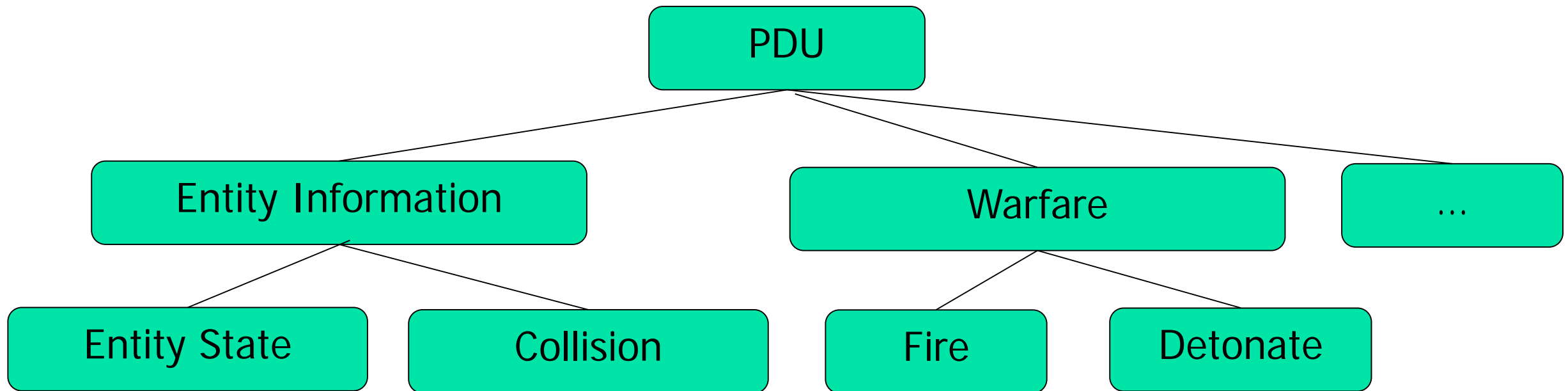
Several dozen different messages (called Protocol Data Units, or PDUs) to describe entity movement, collisions, combat, radio communications, logistics, and more. The Entity State PDU is the most widely used

➢ We use ESPDU when want to inform other hosts of the position of an entity we control—the most common PDU

Entity sends ESPDU with
* Unique ID
* Position (xyz) in standard coordinate system
* Orientation
* What type of vehicle it is
* More....

- Before we can tell an entity "hey you, do that" we need a way to differentiate between entities

- The entity ID is a unique identifier for each simulation object in the world. In DIS this is done via a triplet of three numbers: the Site, Application, and Entity

- These three numbers taken together must be unique for each entity

➢ Example: before the simulation starts we agree, simulation-wide, on the following arbitrary numbers. Here are simple possibilities:

Site	Number
China Lake	42
Norfolk	17
Orlando	23

Application	Number
YoYoDyne M1A2 Simulator	112
ACME UCAV Simulator	417
JCATS	512

Tank Simulator
EID: (42, 112, 18)
EID: (42, 112, 19)

YoYoDyne simulator (112) at China Lake (42) controls two distinct tanks, 18 and 19

Helicopter
EID: (17, 417, 18)

ACME (417) simulator at Norfolk (17) controls a Helicopter (entity 18)

➢ We may need some other information. What if we want to draw this entity? We need to know what this is—a tank, a helicopter, a ship?

➢ This is done via something called the *Entity Type*, which in turn depends on a SISO document called the "Enumeration and Bit Encoded Values" (EBV).

➢ The EBV document is a long listing of standardized record values that lets us identify military hardware

```
{
  Kind: 1 (entity)
  Domain: 1 (Land)
  Country: 225 (US)
  Category: 1 (Tank)
  Subcategory:1 (M1)
  Specific: 6  (M1A2)
}
```
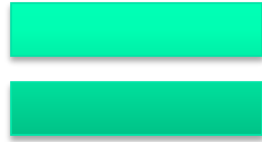

Whenever a DIS message has an entity type record with the above settings we know it's referring to an M1A2 tank. What the numbers are doesn't matter, as long as *all* participants agree on what they mean. SISO maintains this list of arbitrary numbers, called the EBV document

Kind	Domain	Country	Cat	Scat	Specific
1	1	225			

11 M125 81-mm mortar carrier
            1 M125A1
            2 M125A2

12 Cadillac Gage V-600 Armored Car

13 Cadillac Gage LAV Assault Gun LAV-105

15 LAV-105

16 Chrysler MAC-1 armored car

17 Cadillac Gage Commando Scout

18 Cadillac Gage Commando V-300
            1 V-300 w/ TOW
            2 V-300 w/ 81-mm mortar
            3 V-300 ambulance

➤ The EBV document has gone through many versions as new hardware has been added. An important checkpoint for consistency: all participants in the simulation *should* agree on the version of the EBV document used.

➤ In reality this is hard; some simulations have not been updated to reflect new EBV documents, and sometimes those implementing a simulation simply make up numbers, and *no* simulation implements all entities in the EBV.

➤ You may need a gateway such as Joint Simulation Bus (JBUS) to act as a "shim" connection between simulations and change entity type values to match what is expected. The gateway can rewrite the entity type values in the PDUs to force them to match expectations.

➤ The EBV enumerations are also often used in HLA RPR-FOM and in TENA.

➢ What does it mean if we say an entity is at (x, y, z)? This has no meaning without a coordinate system. We need to agree on one, and where the origin is

➢ DIS chose to use a Cartesian, geocentric coordinate system because

It's easy to convert from that to other coordinate systems, such as geodetic (latitude, longitude, altitude) or military Systems such as MGRS
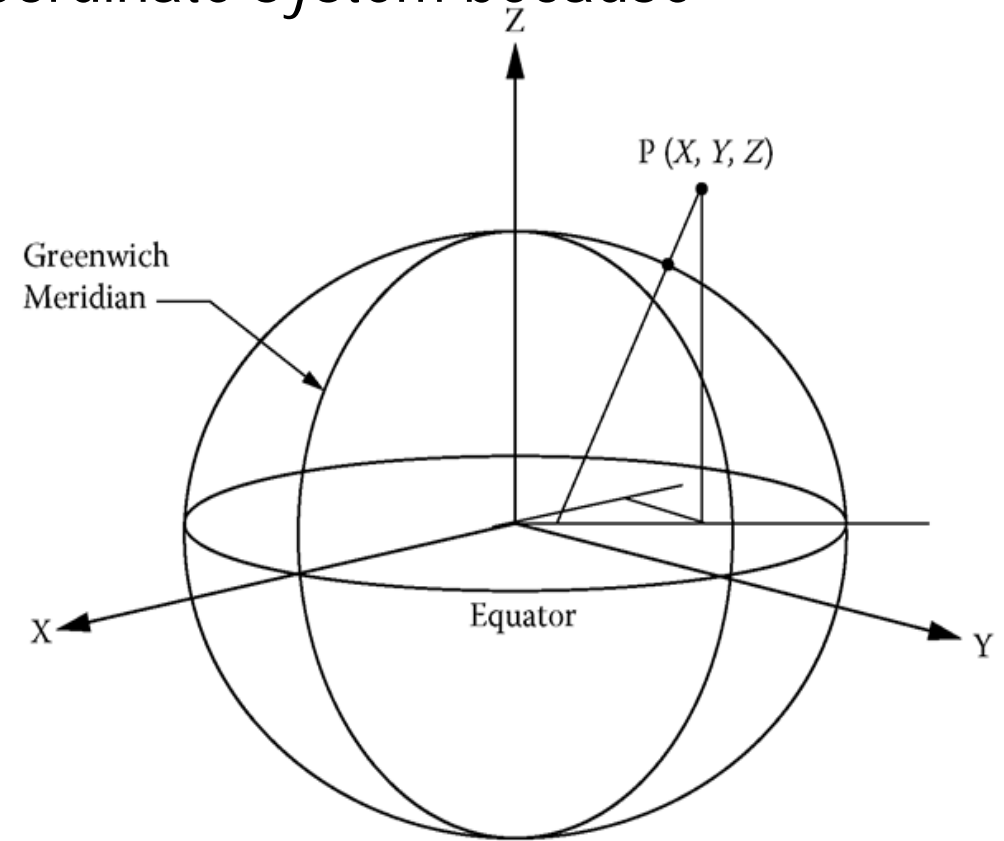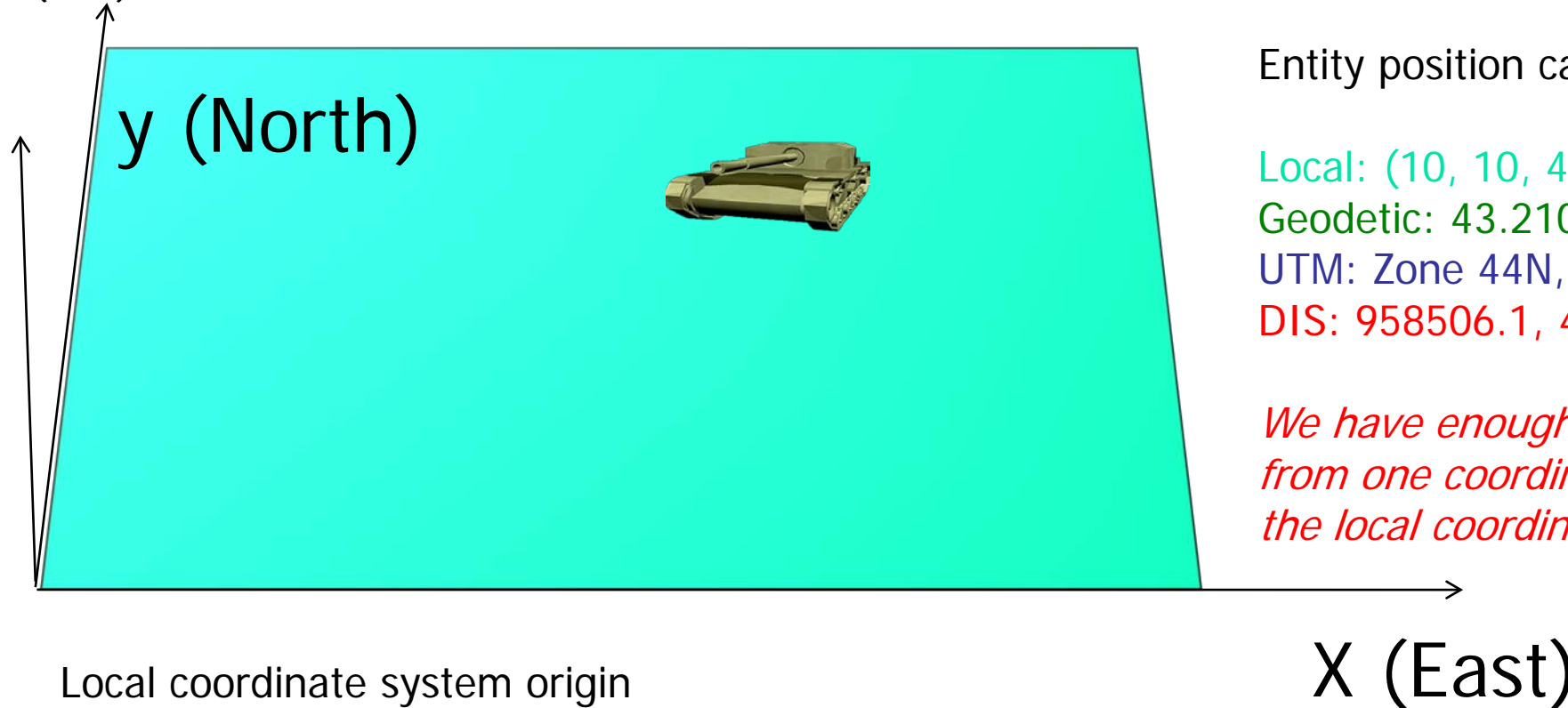
➤ Very often simulations set up a local, flat-plane coordinate system at a point tangent to a point on the earth's surface, and use that for local physics and movement. When the ESPDU is actually sent the local coordinates are changed back to the global coordinate system. The SEDRIS SRM package can do the math

➤ Nobody will ever agree on which way the local coordinate system axes should point for all simulations
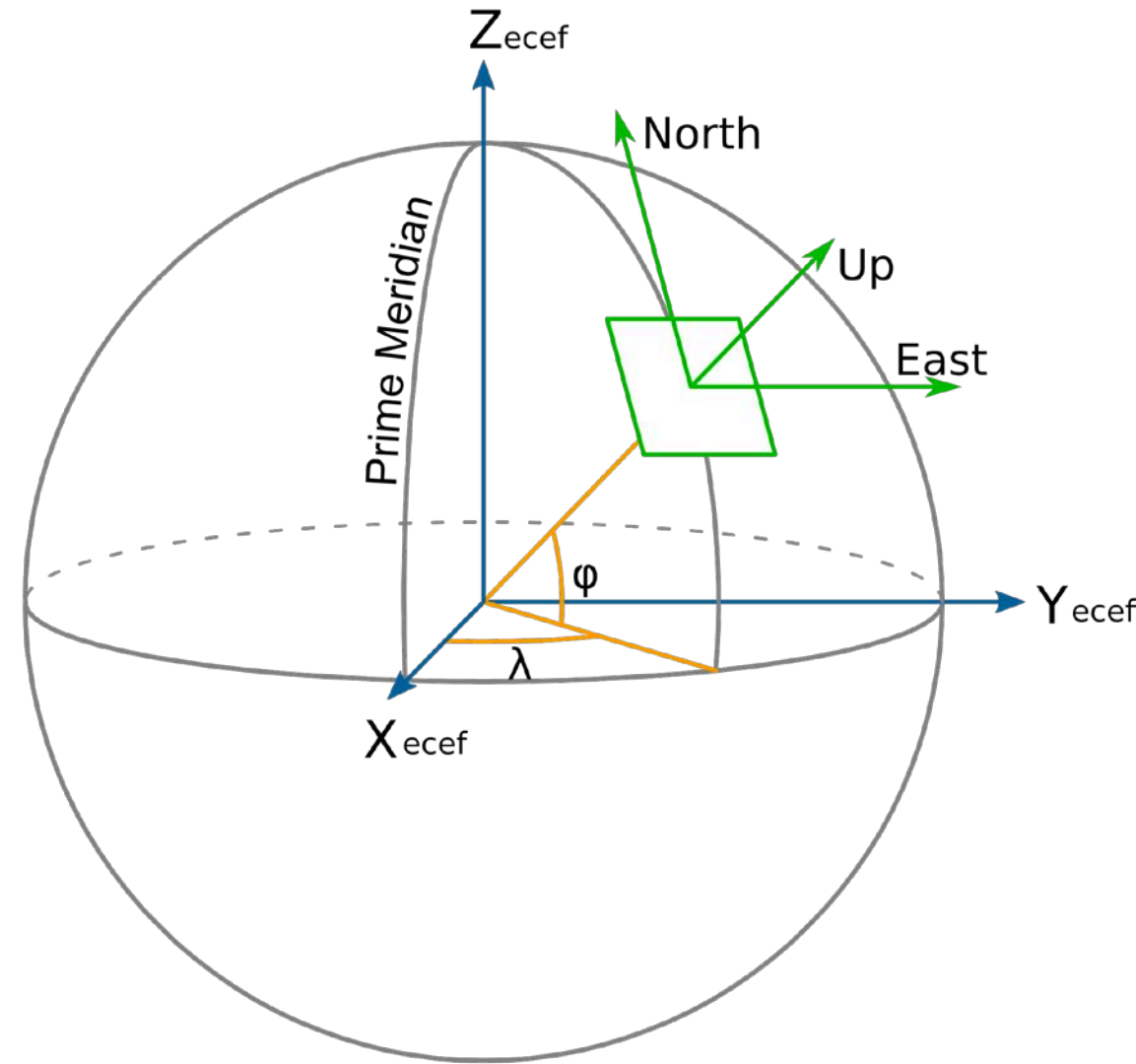
A local Cartesian coordinate system origin is set at {lat, lon, alt}. The simulation does all its calculations and movement in this coordinate system, because it's convenient. Before sending entity information out in a DIS ESPDU, convert from local to geocentric (DIS) coordinates

y (North)

Entity position can be expressed in:

Local: (10, 10, 4)
Geodetic: 43.21001 N, 78.12012W, 124
UTM: Zone 44N, 266061E, 4788172N, 124M
DIS: 958506.1, 455637.2, 4344627.4

We have enough information to convert from one coordinate system to another if we know the local coordinate system origin!

X (East)

Local coordinate system origin
At lat 43.21, lon 78.12, alt 120m, WGS 84

- If we're running a visual simulation, how often do we need to send ESPDUs?
- If running at frame rate, maybe about every 1/30th of a second.
  - If we have 500 entities in a simulation, this can work out to 15,000 UDP messages per second. If you're doing other computation on the host this will tax your CPU and network
- Do we *really* need to send that fast? If we know how fast and in what direction an entity is moving, we can dead reckon in between receiving an ESPDU and draw our entity there.
  - Sure, we're lying to the user. Got a problem with that? If our DR is wrong, we just correct our position stealthily when we get better info on the next packet. The user won't know the difference, probably.
  - Due to latency all simulation participants are a little out of sync anyway

DIS: Dead Reckoning

DR updates

Time

ESPDU Updates
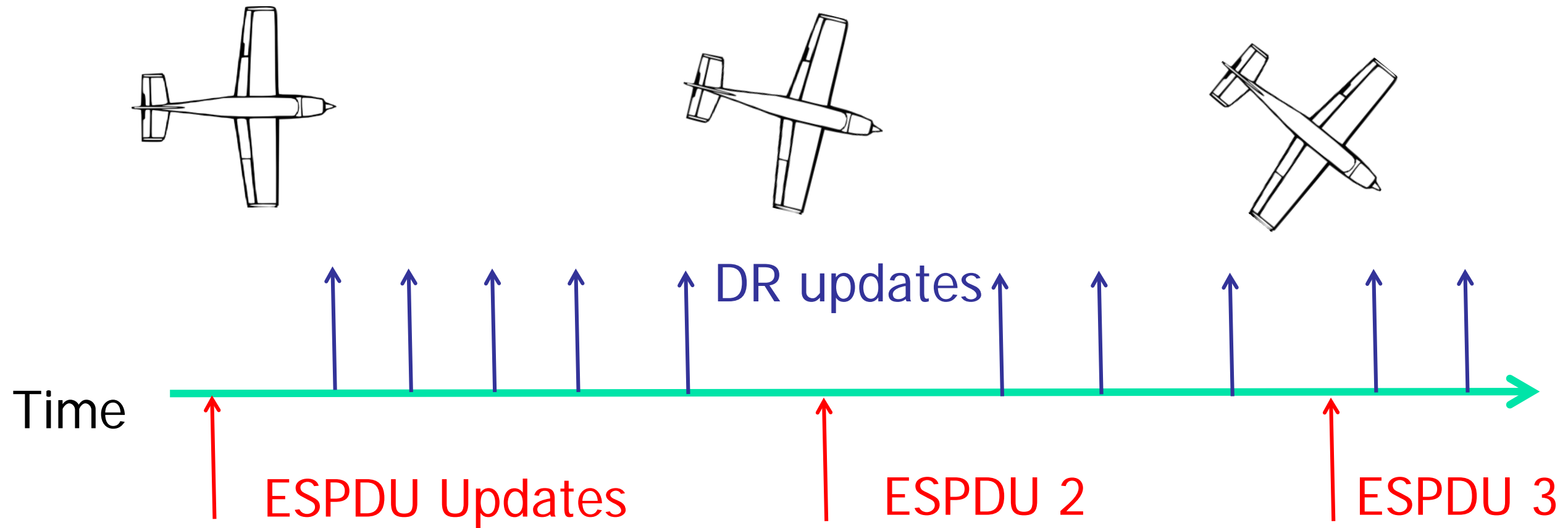
ESPDU 2

ESPDU 3

Aircraft position & orientation updated by dead reckoning updates between ESPDU messages

➢ What DR algorithm is best? (… wait for it…)

➢ It depends! In some situations we might want to include acceleration or angular velocity, but not in others. The ESPDU sender specifies what DR algorithm to use

➢ The sender can also perform its own DR to determine what the recipients are seeing. If the sender decides the clients are probably wrong in their guess about where the entity is, it can issue another ESPDU with better location information

- If we're running a DIS simulation, how do we learn about all the other entities in the world?
  - One design choice DIS could have chosen is to use a server. Instead, the designers chose for DIS to be peer-to-peer; there is no central server, and hosts talk to one another directly
- In DIS all we have to do is listen for ESPDUs. The ESPDUs contain what we need to know: entity type, location, velocity, orientation. This is simple and avoids a single point of failure, and makes configuration easy
  - There are also Simulation Management PDUs for announcing arrival, removal, etc.
- To make this work, every entity must periodically send a time-stamped ESPDU even if its state hasn't changed. This is called a *heartbeat.* Usually entities must send an ESPDU at least once every five seconds

➢ One of the oddities of UDP is that UDP packets may be *duplicated* during TCP/IP routing; we send a single packet, but two might arrive nevertheless.

➢ UDP packets may also arrive *out of order*. We send packets in order A, B, C, but they arrive in order C, A, B
  - This creates problems for position updates! How to handle? ....

➢ DIS includes a *timestamp* field to detect these kinds of problems
  - The field typically represents time since the top of the hour (as common practice)
  - If the next packet we receive has a timestamp before the last packet we processed, we can discard it; it's old information that has been obviated by new data
  - Time coordination between hosts useful but not required
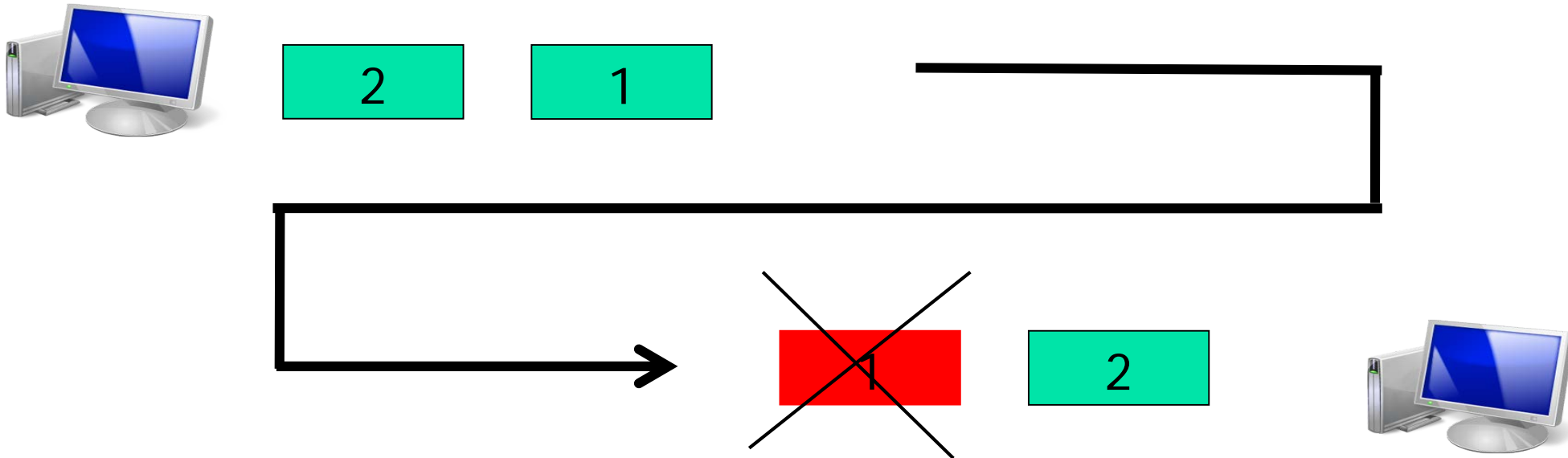
Host A sends 1 and then 2. Host B receives 2, and then 1. We should discard 1, because we know it's older than 2, due to its older timestamp.

➢ So far we've looked at an ESPDU, which contains

- ■ Entity ID
- ■ Entity Type
- ■ Position, orientation, velocity, etc
- ■ Specifies a DR algorithm for the receiver to use
- ■ Timestamp

➢ We've also seen that we need to agree upon a coordinate system, and agree on the enumerations that describe things like entity type.

➢ DIS doesn't have an API. This seems strange to people coming from HLA or TENA, but reflects common practice in networking protocols

- The standardized part is the format of the messages on the wire. The standard is silent about how to create or receive those messages

- Different DIS vendors have different APIs, *but all produce the same format messages. This is in contrast to HLA, which has a standard API, but is silent about the format of messages on the wire.* As a result, different HLA RTI vendors usually use different message formats for exchanging information

- TENA standardizes the API, and there is a single approved implementation of the RTI equivalent; this sidesteps the wire standard problem because there is only one approved equivalent of the RTI

JCATS

ACME Sim

HLA RTI API

HLA RTI API

RTI Vendor-Specific Format Messages

Network

HLA RTI API

ONE-SAF
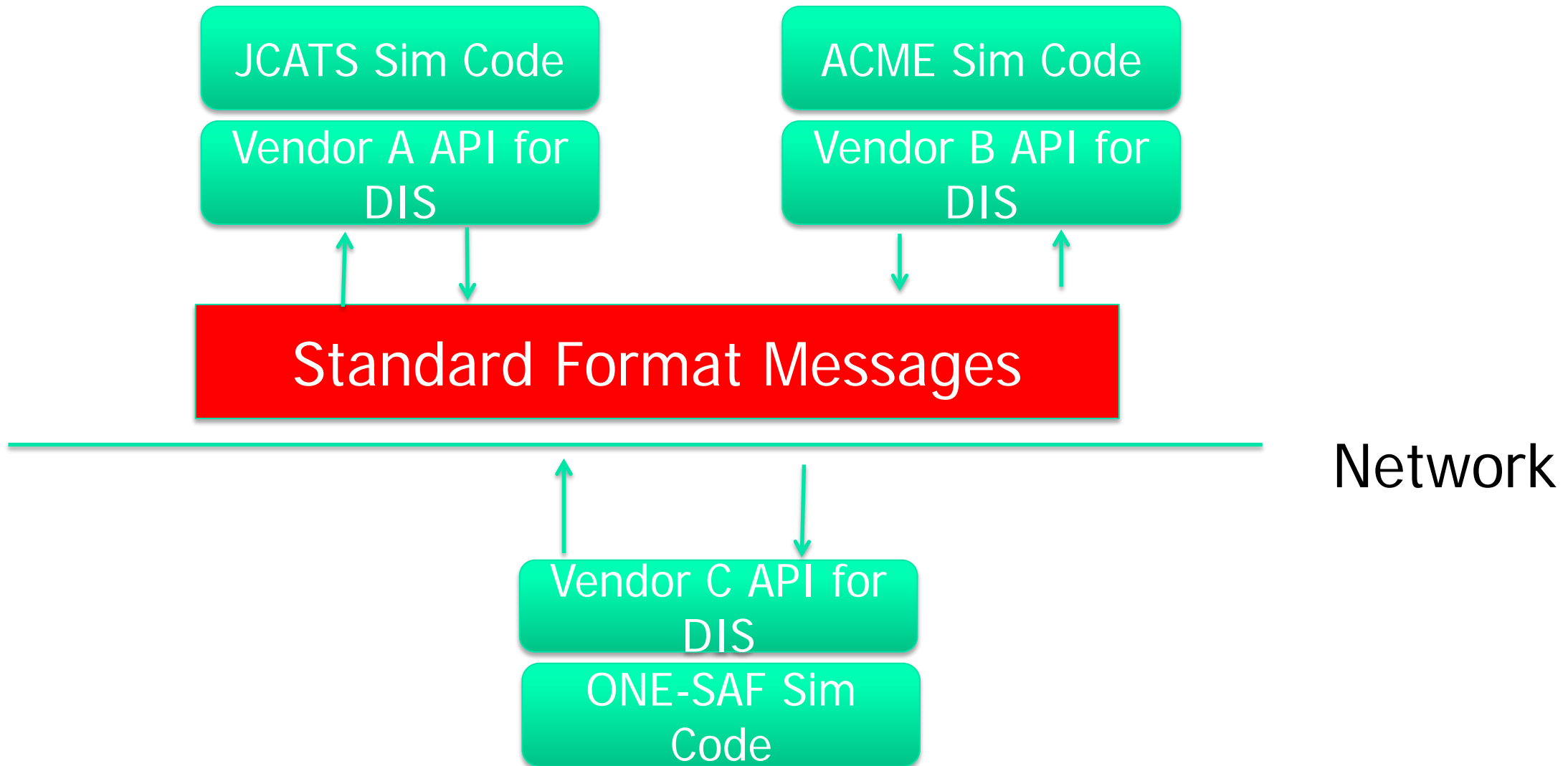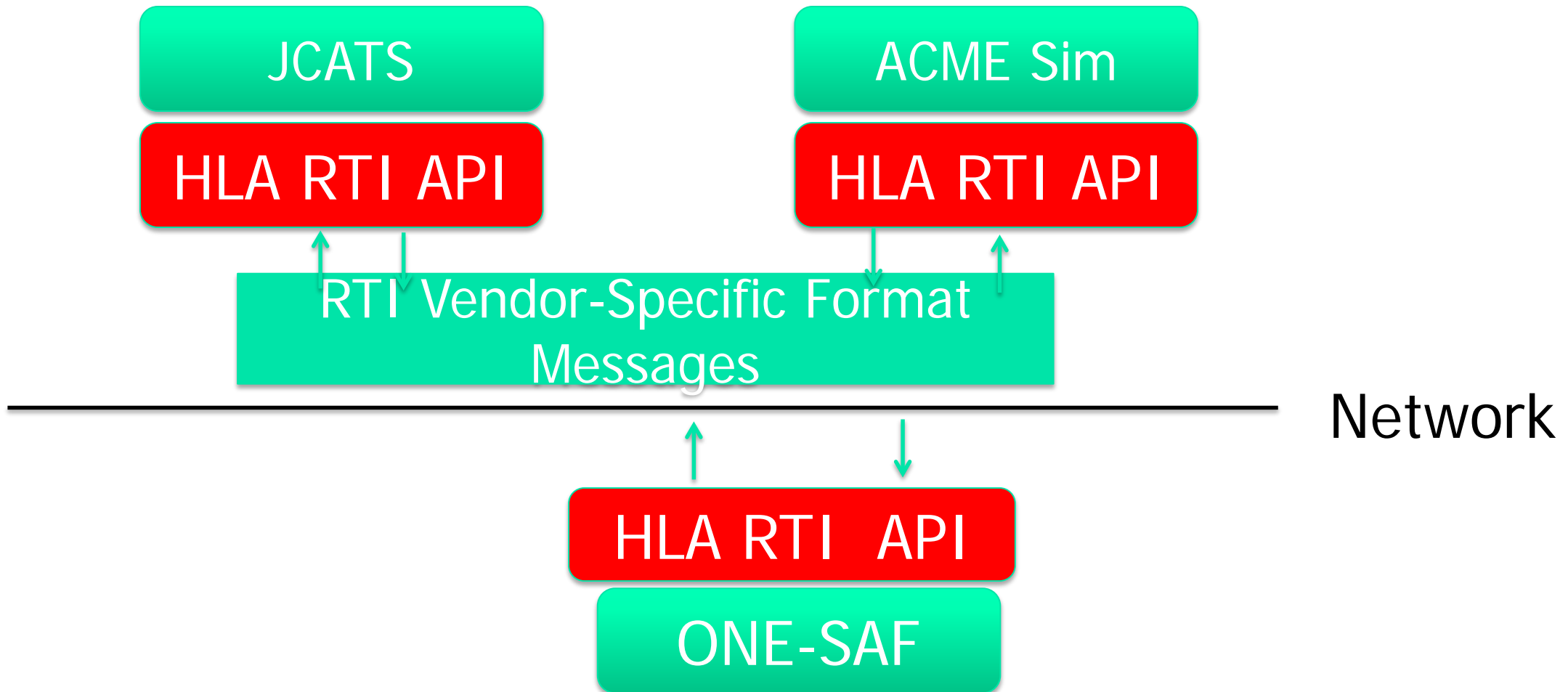
While the API is standard, implementations of the HLA RTI API from different vendors are allowed to produce messages in different formats.

➢ The implications of this are that while HLA has a standardized API, RTIs from different vendors can't typically talk directly to each other. This makes changing vendors easy, but makes getting RTIs from different vendors talking to each other hard--you need to use a gateway

➢ DIS in contrast makes changing vendors hard (since it involves changing the API your simulation code uses) but talking between vendors easy (since all the messages on the wire are in the same format)

■ The lack of an API can help when using unusual languages, such as Objective-C, C#, Python, and Javascript. Since there's no API, just make one up. As long as they produce standard messages, it doesn't affect anyone else

➢ Remember, all this information is being sent in binary format in (typically) a UDP packet

➢ The *exact* format that an ESPDU must have on the wire is specified in the DIS standard. This includes byte order

➢ For example, the EntityID field starts 12 bytes into the ESPDU message, is in the order (site, application, entity), and each field entry is 16 bits long, in network byte order, and unsigned
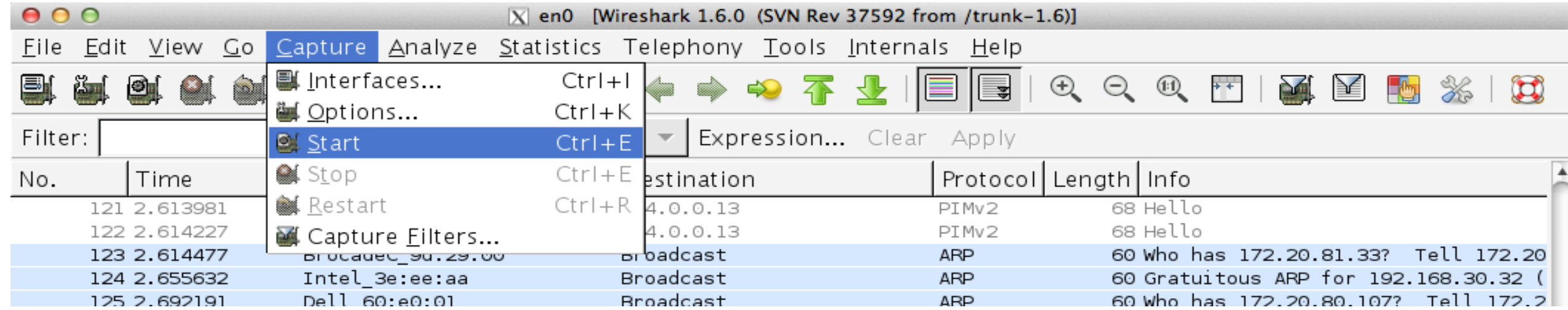
Table 134—Entity State PDU

Field size (bits)	Entity State PDU fields	
96	PDU Header	Protocol Version—8-bit enumeration
		Exercise ID—8-bit unsigned integer
		PDU Type—8-bit enumeration = 1
		Protocol Family—8-bit enumeration = 1
		Timestamp—32-bit unsigned integer
		Length—16-bit unsigned integer
		PDU Status—8-bit record
		Padding—8 bits unused
48	Entity ID	Site Number—16-bit unsigned integer
		Application Number—16-bit unsigned integer
		Entity Number—16-bit unsigned integer

- What do ESPDUs look like? We can examine them on the network with a free tool called Wireshark, which can decode DIS packets
  - http://www.wireshark.org

- Remember, it's the format of the messages on the wire that count. The DIS standard specifies the exact format of binary messages, and any tool that produces or consumes those messages is fine with DIS. How you create them is none of DIS's business

Wireshark: Capture Packets

- ➢ Format
  - ■ We know what information we want to send: entity type, entity ID, position, orientation, etc.
  - ■ We know what coordinate system we want to use
  - ■ We know where to find arbitrary, agreed-upon enumeration identifier values—the EBV document
  - ■ We know some PDU types: entity state PDU, etc.
- ➢ How do we get the information into the format we want on the wire?
- ➢ This is where DIS implementations come in

➢ Where can you get a DIS implementation?

- ■ Write your own (cough cough)

- ■ Buy one. There are several commercial implementations and many have excellent support

- ■ Use an open source version—"free as in free puppy"

  - ■ Open-DIS (https://github.com/open-dis - formerly http://open-dis.sourceforge.net)

    - ■ Java, C++, C#, Objective-C, JavaScript, Python

  - ■ KDIS (http://sourceforge.net/projects/kdis)

    - ■ C++

  - ■ Aquarius (http://sourceforge.net/projects/aquariusdispdu)

    - ■ C++

  - ■ JDIS (http://sourceforge.net/projects/jdis)

    - ■ Java

➢ Remember, DIS has no official API. Every implementation is different. This example will use the Open-DIS API codebase. Implementation available at https://github.com/open-dis

➢ Source code for this example is available at http://www.movesinstitute.org/DIS

➢ The example contains a lot of supporting libraries for other things. Ignore all that for now.

➢ All code is BSD open source license; nonviral, use any way you see fit

```java
public void sendEspdus()
{
    try
    {
        EntityStatePdu espdu = new EntityStatePdu();

        // Set the entity ID, the unique identifier for an entity in the world
        espdu.getEntityID().setSite(CHINA_LAKE);
        espdu.getEntityID().setApplication(NPS);
        espdu.getEntityID().setSite(1);

        // Set this up as a Russian BMP-1
        EntityType type = espdu.getEntityType();
        type.setEntityKind((short)1);    // vehicle
        type.setDomain((short)1);        // land
        type.setCountry(222);            // Commie russians
        type.setCategory((short)2);      // AFV
        type.setSubcategory((short)1);   // BMP
        type.setSpec((short)1);          // basic BMP

        // Set position
        espdu.getEntityLocation().setX(1000.0);
        espdu.getEntityLocation().setY(2000.0);
        espdu.getEntityLocation().setZ(3000.0);

        for(int idx = 0; idx < 100; idx++)
        {
            byte data[] = espdu.marshalWithDisAbsoluteTimestamp();
            DatagramPacket packet = new DatagramPacket(data, data.length, destinationAddress, port );
            socket.send(packet);
            Thread.sleep(1000);
        }

    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
```

Can use HTML5 Browser geolocation and Javascript to send DIS from a web page

```javascript
// Set location. We get an object with properties x, y, and z back
// from conversion.
var disCoordinates = conversion.getXYZfromLatLonAltDegrees(latLonAlt);
espdu.entityLocation.x = disCoordinates.x;
espdu.entityLocation.y = disCoordinates.y;
espdu.entityLocation.z = disCoordinates.z;

espdu.marking.setMarking("Browser");

var dataBuffer = new ArrayBuffer(1000); // typically 144 bytes, make it big
var outputStream = new dis.OutputStream(dataBuffer);
espdu.encodeToBinaryDIS(outputStream);

// Trim to fit
var trimmedData = dataBuffer.slice(0, outputStream.currentPosition);
websocket.send(trimmedData);
```

DIS: Receive PDUs in Java

```java
public void receivePdus()
{
    try
    {
        PduFactory pduFactory = new PduFactory();

        while(true)
        {
            byte[] buffer = new byte[1500];
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

            socket.receive(packet);

            Pdu aPdu = pduFactory.createPdu(packet.getData());

            int pduType = aPdu.getPduType();

            switch(pduType)
            {
                case 0:
                    System.out.println("Unrecognized PDU type");
                    break;

                case 1:
                    System.out.println("Got Entity State Pdu");
                    break;

                case 2:
                    System.out.println("Got Fire PDU");
                    break;

                default:
                    System.out.println("got some type of PDU");
                    break;

            }

        }
    }
}
```

➢ There are similar idioms for other languages such as C++, Objective-C (IOS/MacOS), Javascript, Python, C# (Windows phone, Unity 3D)

➢ Note that this requires that you do a bit of socket programming, which TENA and HLA hide from you. Socket programming isn't *that* bad…

➢ DISWebGateway running at

➢ http://track.movesinstitute.org

- Java sender & receiver for DIS

- Can receive native DIS from existing DIS applications

- Web-based map that shows DIS entity locations
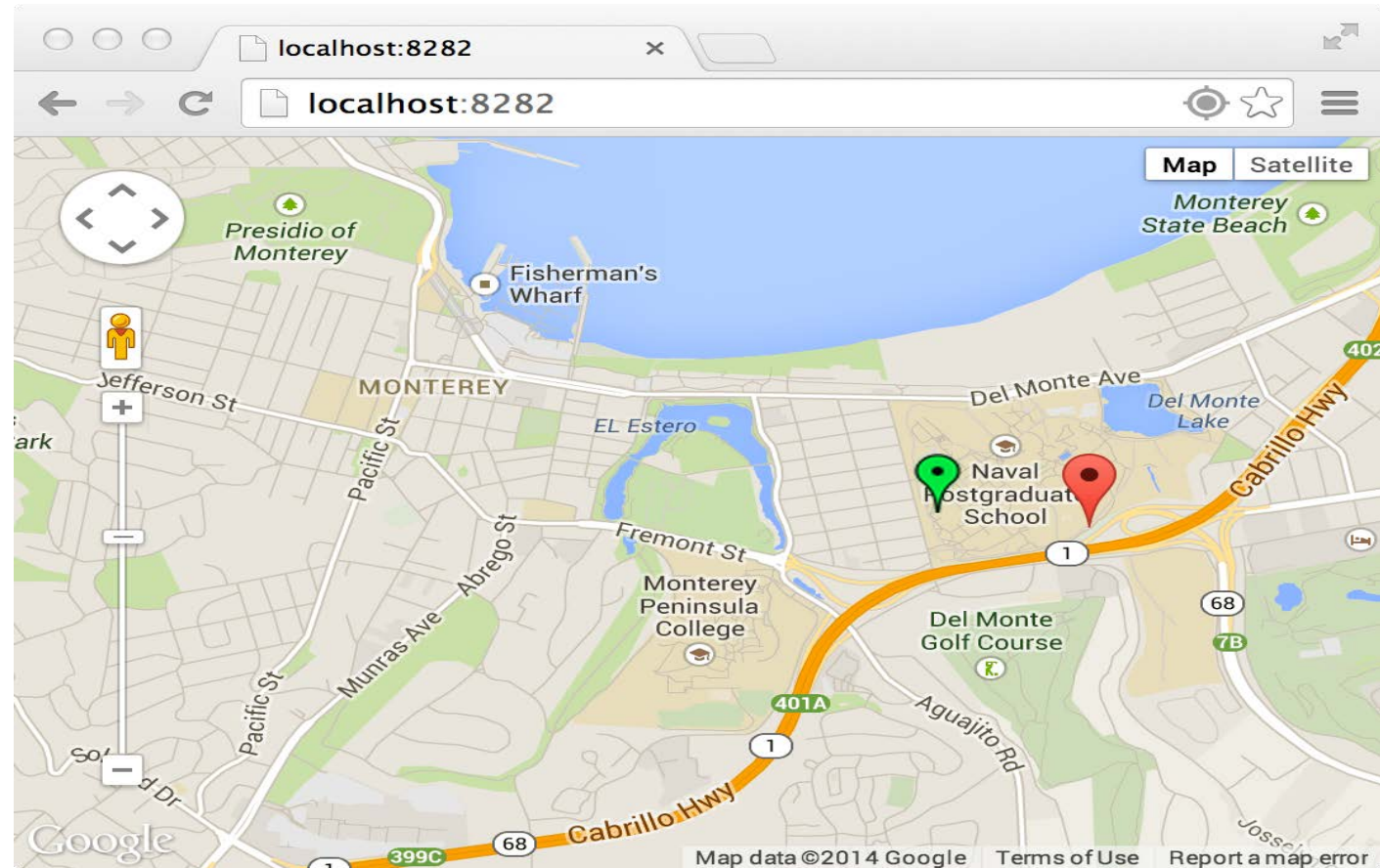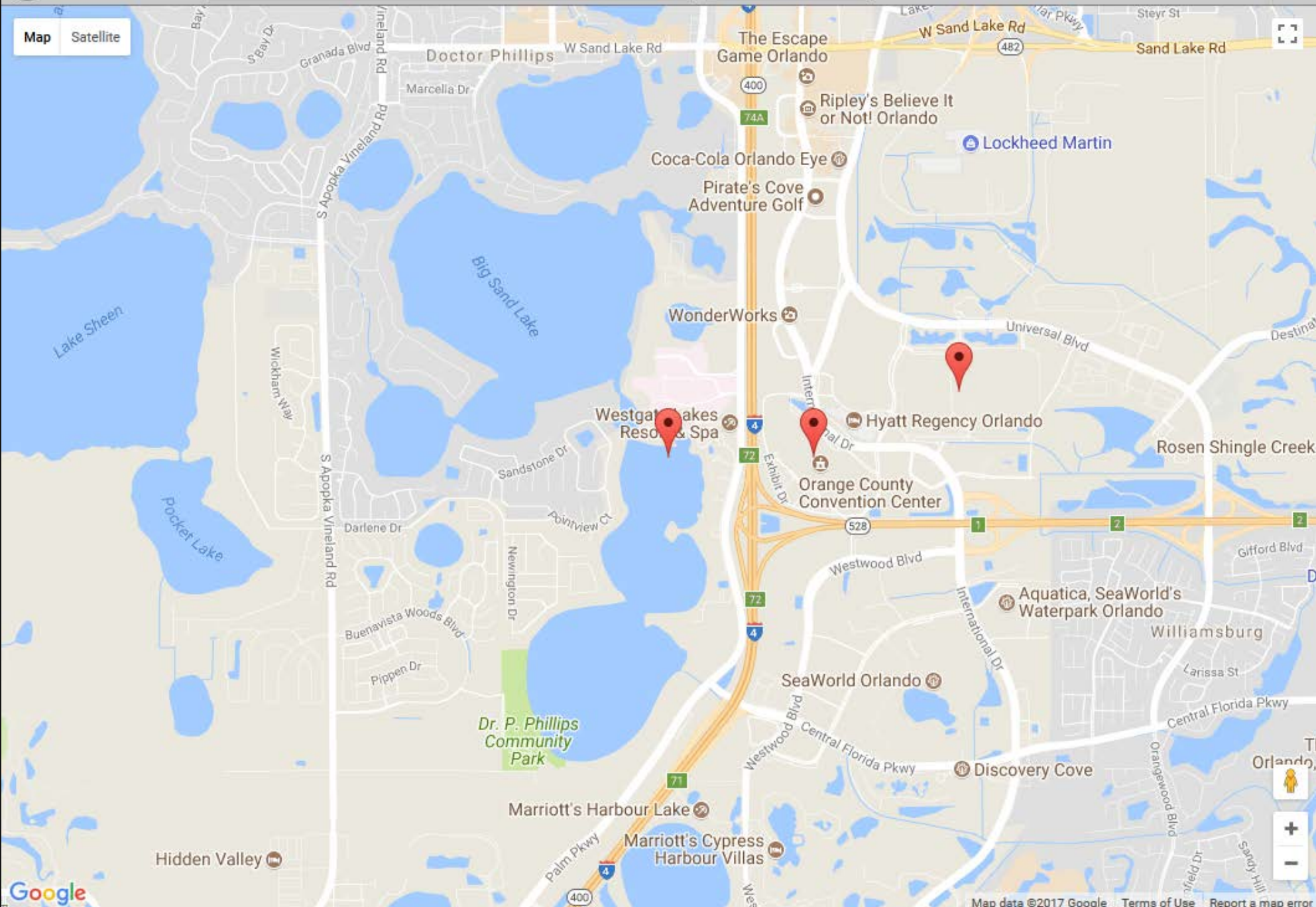
Open Street Map/Open Layers 3 implementation of binary DIS to the web browser
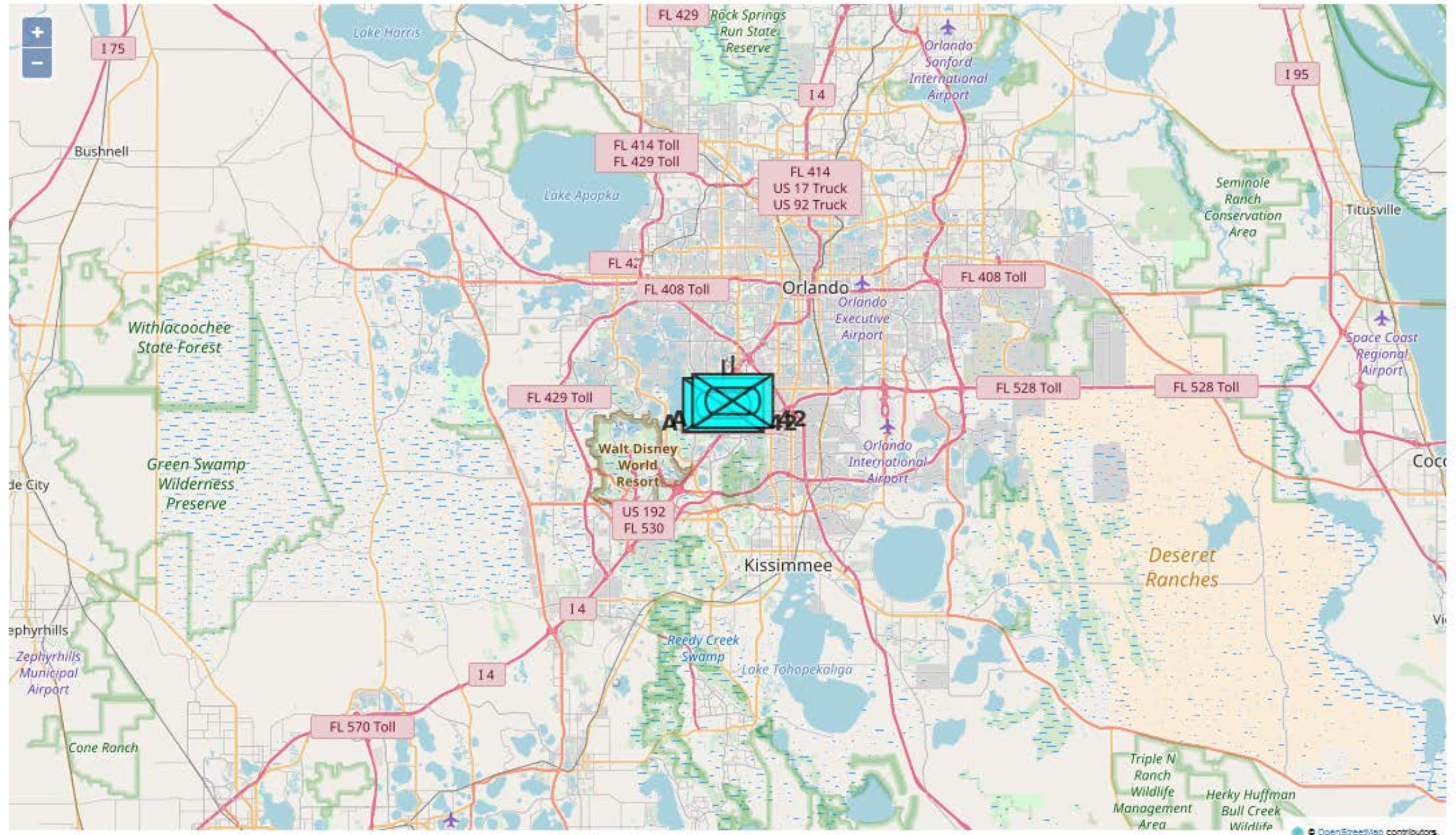
Constructive Simulation DIS Entity

This web page sends Entity State PDUs with the values below. Fill out the form, and click "Submit." The page will send ESDPUs with those values to the web server over a websocket, and the server will transmit them on the local network.

The Javascript in the web page will send the ESPDUs with the frequency indicated (in milliseconds).

Entity Type Information; Russian T-72 by Default
Kind: 1
Domain: 1
Country 222
Category 1
Subcategory 2
Specific 1
Extra 1

Entity ID site and application
Site 48
Application 23

Location
Latitude (Decimal degrees): 28.425
Longitude -81.47
Altitude (meters) 10
28.425
Marking Marking MyEntity

Send Frequency
Send Frequency (ms) 5000
Start Sending PDUs

➢ We've been sending ESPDU messages back and forth, but there are dozens of other sorts of messages. What if we want to shoot at someone? What does this involve?

➢ We can use a Fire PDU, which contains

- The entity ID of the shooter
- The entity ID of the target (if known)
- The type of munition being fired, fuse, quantity, etc. This is very similar to the entity type
- Enough information to compute the path of the munition (if desired)

➢ The Detonation PDU usually follows a Fire PDU. It contains

- Location of detonation, shooter entity ID, target entity ID
- Fuse, munition type, and so on

➢ When a Detonation PDU is received simulations assess and report damage to their own entities, not to others.

- This means simulations are on the honor system for determining damage; thus James T. Kirk can beat the unwinnable Kobayashi Maru scenario.
- Here is a harsh adjective to describe cheating in distributed scenarios: **boring**.
- For military simulations we are much more interested in strengths, vulnerabilities and possibilities that may occur in the real world. Thus cheating is also critically unhelpful.
- Trusted participants using trusted software suites can reduce the risk of cheating.

Fire! I'm entity (17, 23, 42), shooting at entity (123, 7, 12) with a HEAT round from (x, y, z)

Explosion! There's an detonation of a HEAT round at (x', y', z').

All entities now assess the damage to themselves by the Detonation at (x', y', z')

➢ You can also exchange arbitrary data between DIS simulation participants with the DataQuery and Data PDUs.

  ■ Participant sends a DataQuery PDU addressed to another participant

  ■ That participant responds with a Data PDU

➢ The data itself is sent as "fixed variable datums" or "variable data datums". Therefore it's up to you to specify the exact format of these

- ➢ There are many other messages that can be used in DIS
  - Electronic warfare
  - Logistics
  - Directed energy weapons
  - Voice/Intercom
  - Collisions
  - Simulation management
  - Data exchange

- ➢ It's a big topic!
- ➢ But the basics are: a standard format for exchanging state information

➢ **Dead Reckoning (DR) is projection of entity location based on last received timestamp, position and vector-based velocities/accelerations**
- Enables recipients to more accurately estimate entity state in-between PDU updates
- Enables senders to more accurately estimate when to send more-frequent updates

➢ **Smoothing is a recipient-presentation technique for handling dropped packets**
- Avoid sudden jerky jumps in motion, instead interpolate from prior estimate to new state
- No need to improperly distract user with corrective actions when recovery is satisfactory

➢ **Synchronization among multiple players is a blend of capabilities**
- Networking monitoring and management, simulation management, logging, etc.
- Measurable metric: is a "fair fight" taking place among distributed participants?

➤ Dead Reckoning (DR) algorithms use projected trajectory information (such as linear or rotational velocities and accelerations to compute how often network updates need to be sent. Helpful for quiet and intensely active intervals.

➤ Visual smoothing techniques hide when packets are dropped or arrive late. Entity display smoothly interpolates to the correct location and direction, avoiding distracting jumps that do not correctly represent behavior anyway.

➤ Having each individual entity honorably compute whether collisions occurred takes advantage of highest fidelity information with least computation, avoiding expensive time delays and greater inaccuracies of server-based adjudications.

- How can DIS interoperate with HLA or TENA?

- HLA Real-time Platform Reference Federation Object Model (RPR-FOM)
  - Intentionally matches DIS, same entity types, same entity IDs and coordinate system, etc.
  - HLA object model mapping makes transition from DIS to HLA easy and consistent
  - Several gateways to translate between DIS and RPR-FOM. JBUS, AMIE, others
  - Guidance, Rationale and Interoperability Modalities (GRIM) for RPR-FOM standard provides further rules and usage information.
  - https://www.sisostds.org/productspublications/standards/sisostandards.aspx

- TENA has generalized gateway functionality that can map TENA events to DIS and vice versa. It generally uses the same coordinate system, entity types, etc.

- ➢ Websockets are a standard from IETF and W3C. The idea is to provide a direct Javascript-based TCP socket into a web page without having to use AJAX polling techniques. Widespread browser support

- ➢ Javascript is a widely used language for dynamic web content
  - WebGL is Javascript binding for OpenGL which allows us to use accelerated 3D graphics inside the web page
  - WebGL can be the substrate for higher level graphics standards such as X3D

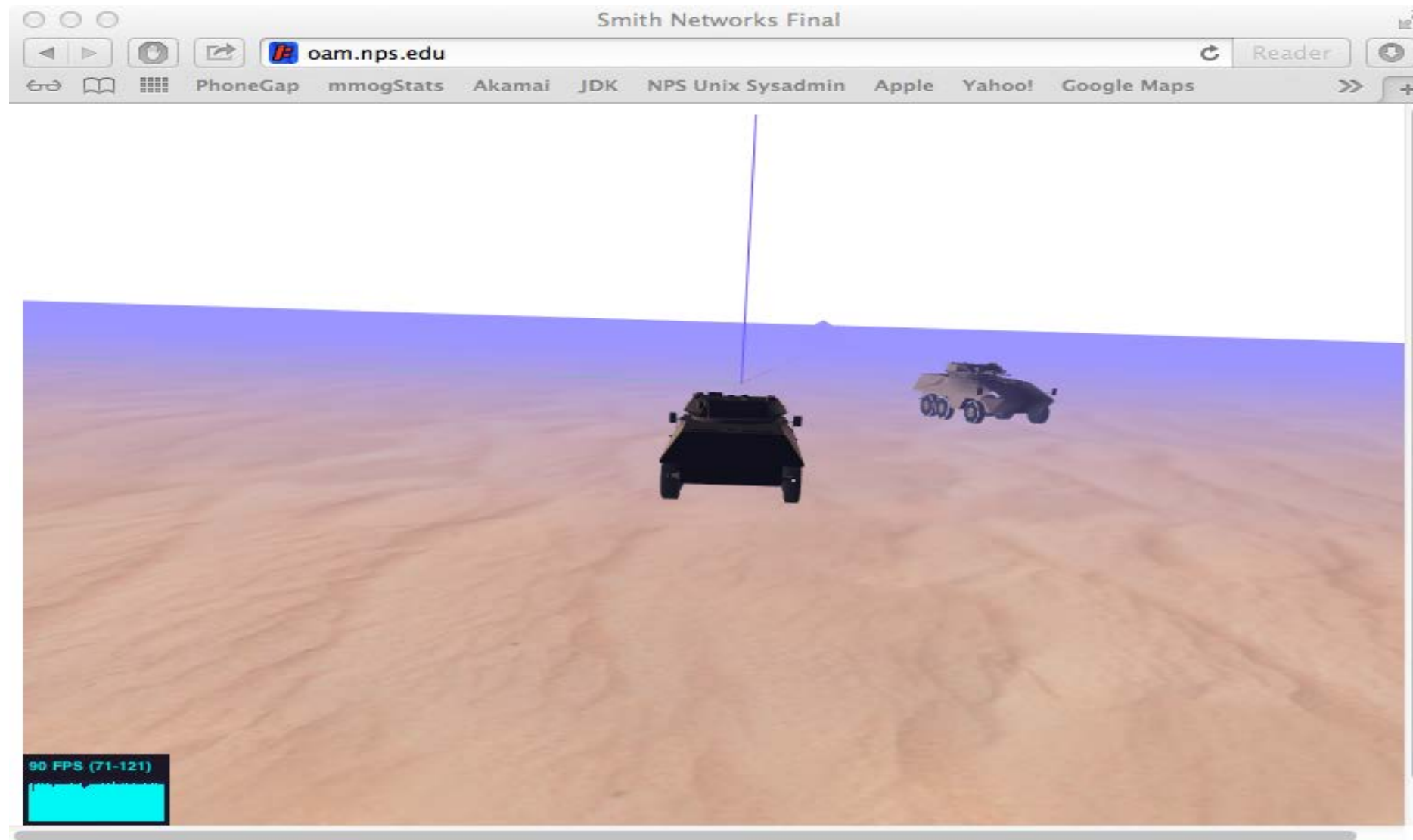- ➢ Put all three together and you can implement a networked virtual environment in a web page

Web Server

Network with conventional binary-format DIS packets

Web pages with Javascript WebGL scene updated by DIS over a Websocket

- Excellent combinations of WebGL/X3D, WebSockets, and fast JavaScript in the web browser have emerged in recent years
  - Open-DIS library can send DIS PDUs directly into a web browser

- SISO WebLVC Product Development Group (PDG)
  - https://www.sisostds.org/StandardsActivities/DevelopmentGroups/WebLVCPDG.aspx
- One of many examples: Virtual World Framework
  - http://virtualworldframework.com and https://en.wikipedia.org/wiki/Virtual_world_framework

- Active research topic… but no wide-scale "Ready Player One" arenas for DIS, yet

➢ DIS applications exchange state information among distributed set of players.

➢ Defines syntax and semantics for a series of binary-formatted messages, with each packet's bytes, data representation and functionality exactly defined.

➢ Different software APIs can implement the same "over the wire" data standard.

➢ Applications focused on large-scale, high-fidelity, virtual / constructive sims.

➢ Common concepts: entity types, entity IDs, heartbeats, coordinate systems.

- SISO: http://sisostds.org

- SISO DIS Protocol Support Group:
  http://discussions.sisostds.org/topiclistview.aspx?fid=32

- Open-DIS: https://github.com/open-dis

- SEDRIS SRM: http://sedris.org

- Kdis: http://kdis.sourceforge.net

- Wireshark: http://wireshark.org

- WebGL: http://www.khronos.org/webgl

- X3D: http://www.web3d.org and http://x3dgraphics.com/slidesets

- WebLVC: https://www.sisostds.org/StandardsActivities/DevelopmentGroups/WebLVCPDG.aspx

- WebSockets: http://tools.ietf.org/html/rfc6455 , http://www.w3.org/TR/websockets