



**Dr. Richard Fujimoto**  
Georgia Institute of Technology  
Designer of HLA Time Management



**Dr. Kalyan Perumalla**  
Oak Ridge National Laboratory  
Creator of time management tutorial slides

# Tutorial Handling Time Management Under the High Level Architecture

*Kalyan Perumalla, Ph.D.*

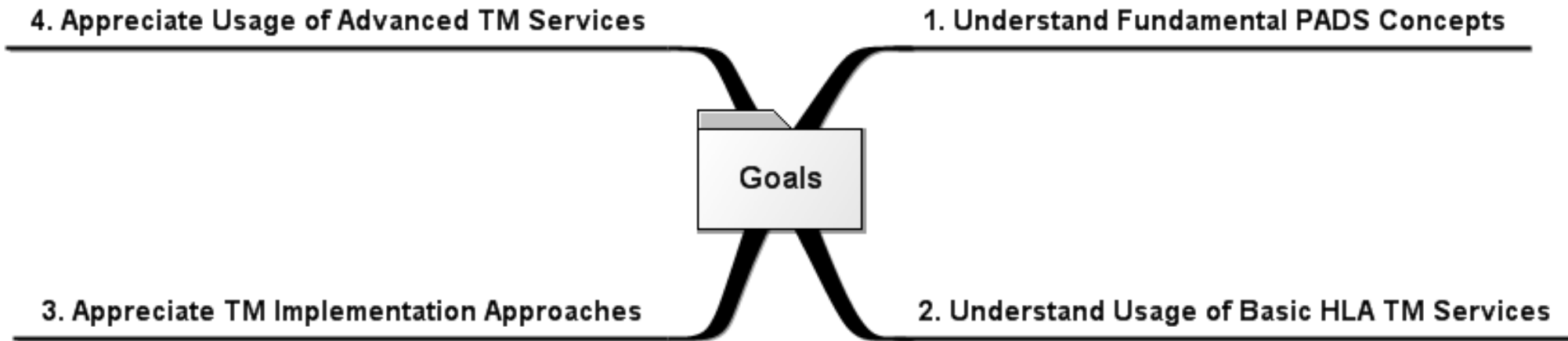
Senior Researcher, Oak Ridge National Laboratory  
Adjunct Faculty Member, Georgia Inst. of Technology

`perumallaks@ornl.gov`

`www.ornl.gov/~2ip`

*December 2007*

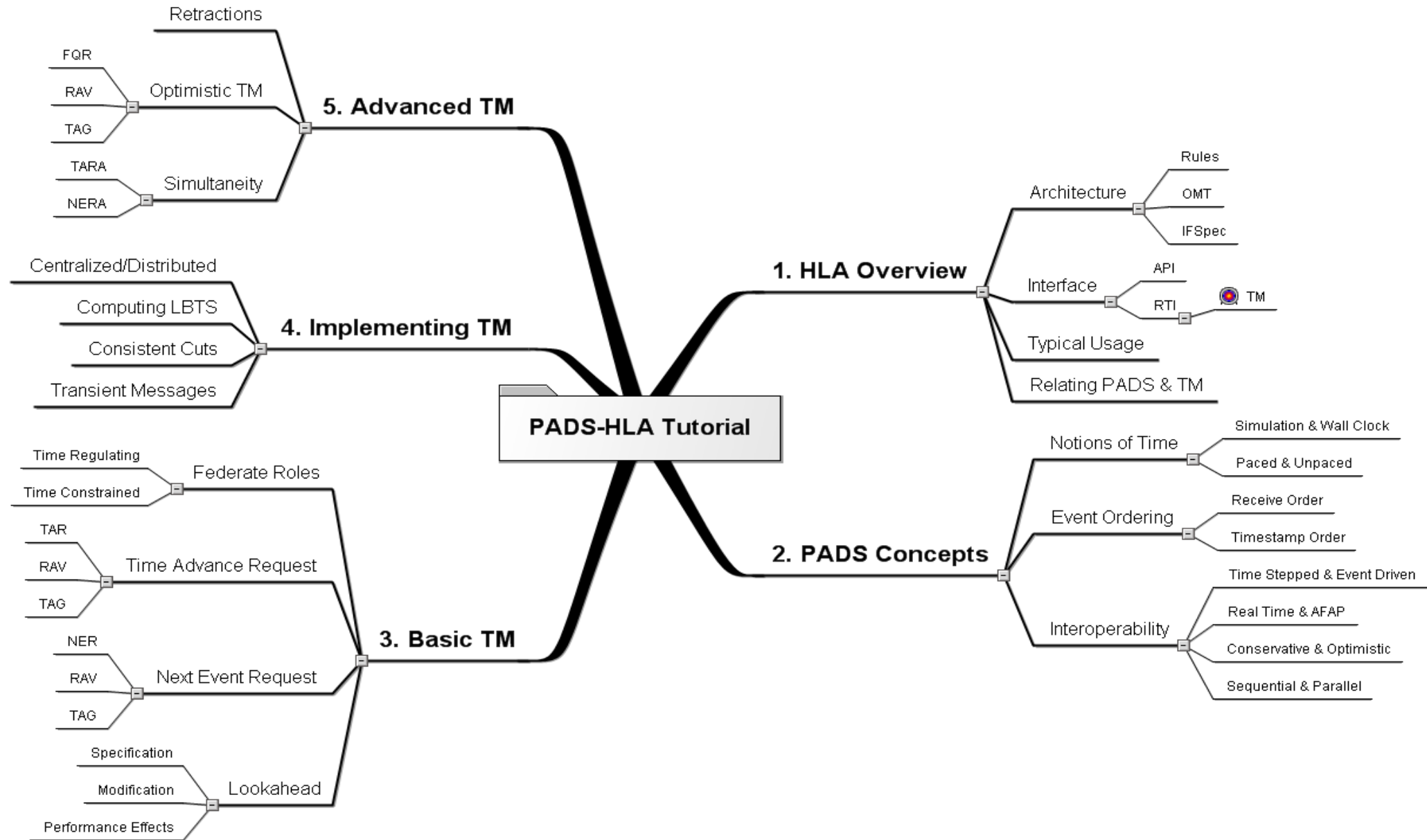
# PADS-HLA Tutorial Goals



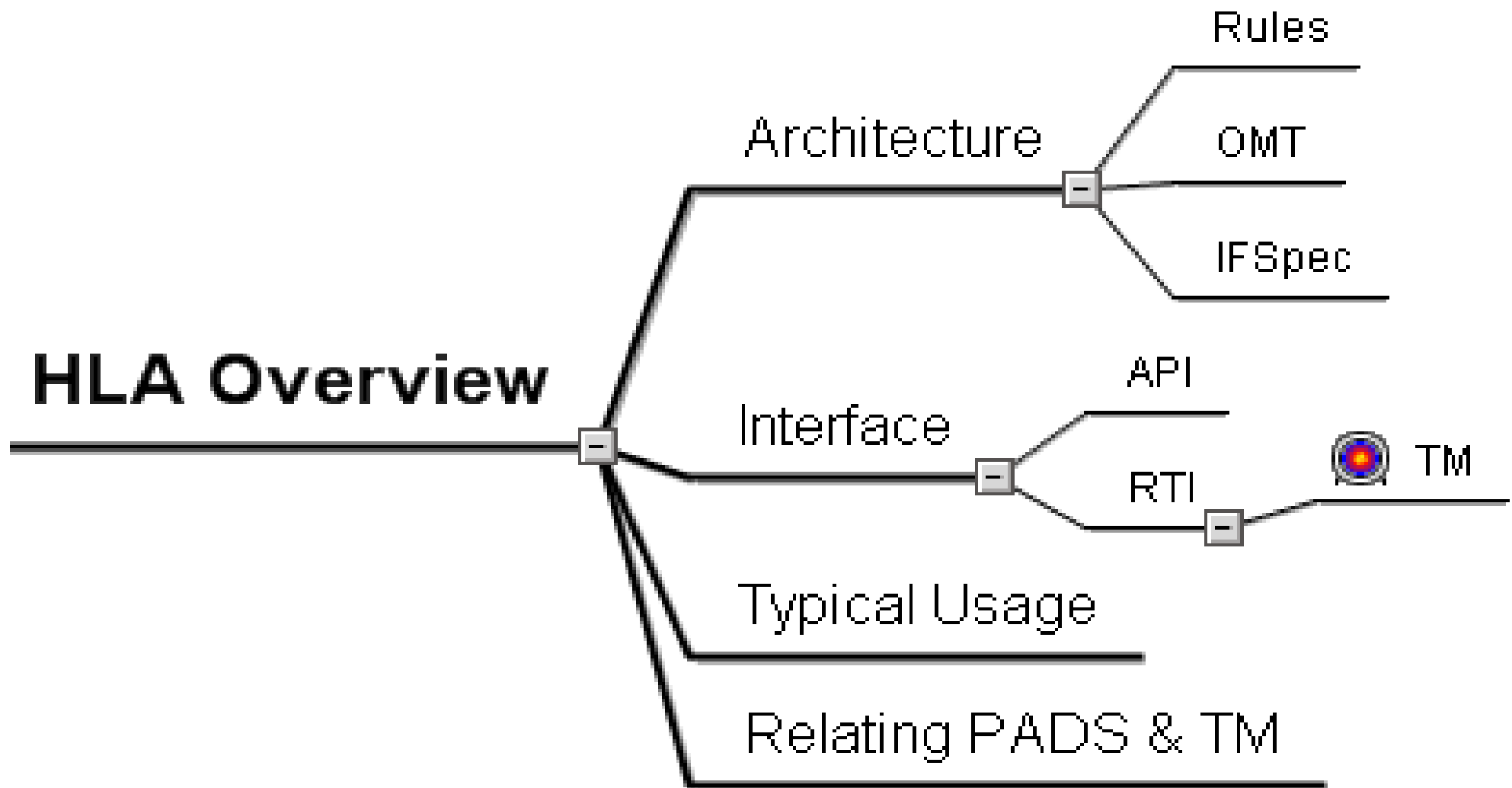
Note: Here, PADS represents the concepts, *not* the specific IEEE/ACM/SCS conference on PADS!

PADS = Parallel and Distributed Simulation  
HLA = High Level Architecture  
TM = Time Management

# Outline



# 1. HLA Overview



# HLA – Architecture

- Based on a composable “system of systems” approach
  - No single simulation can satisfy all user needs
  - Support interoperability and reuse among DoD simulations
- *Federations* of simulations
  - Pure software simulations
  - Human-in-the-loop simulations (virtual simulators)
  - Live components (e.g., instrumented weapon systems)

The HLA consists of

- **Rules** that simulations (federates) must follow to achieve proper interaction during a federation execution
- **Object Model Template (OMT)** defines the format for specifying the set of common objects used by a federation (federation object model), their attributes, and relationships among them
- **Interface Specification (IFSpec)** provides interface to the *Run-Time Infrastructure (RTI)*, that ties together federates during model execution

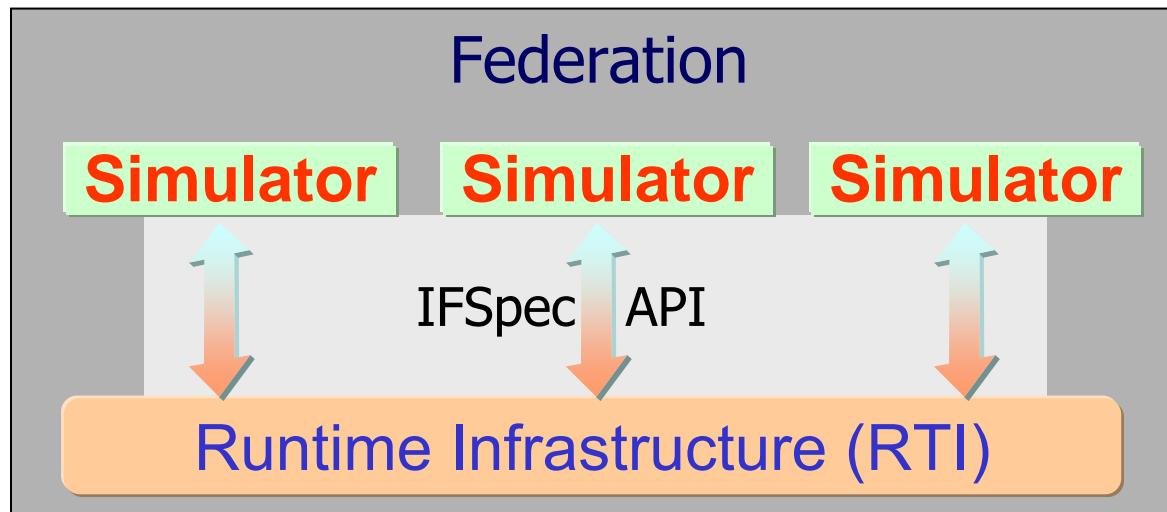
# Interface – Integrating Individual Simulators

## Individual/autonomous simulator

- Synonymous with *Federate* in HLA terminology
- Conforms to, and uses, HLA Interface Specification (IFSpec)

## RTI implements interface services

- Federation setup and teardown
- Data exchange among federates
- Synchronization for timestamp-ordered processing



# Interface – Services

Category	Functionality
Federation Management	Create and delete federation executions join and resign federation executions control checkpoint, pause, resume, restart
Declaration Management	Establish intent to publish and subscribe to object attributes and interactions
Object Management	Create and delete object instances Control attribute & interaction publication Create and delete object reflections
Ownership Management	Transfer ownership of object attributes
Time Management	Coordinate the advance of logical time and its relationship to real time
Data Distribution Management	Supports efficient routing of data



# Typical Usage – Federate Execution

## Initialize federation

- Create Federation Execution (Federation Mgt)
- Join Federation Execution (Federation Mgt)

## Declare objects of common interest among federates

- Publish Object Class (Declaration Mgt)
- Subscribe Object Class Attribute (Declaration Mgt)

## Exchange information

- Update/Reflect Attribute Values (Object Mgt)
- Send/Receive Interaction (Object Mgt)
- Time Advance Request, Time Advance Grant (Time Mgt)
- Request Attribute Ownership Assumption (Ownership Mgt)
- Modify Region (Data Distribution Mgt)

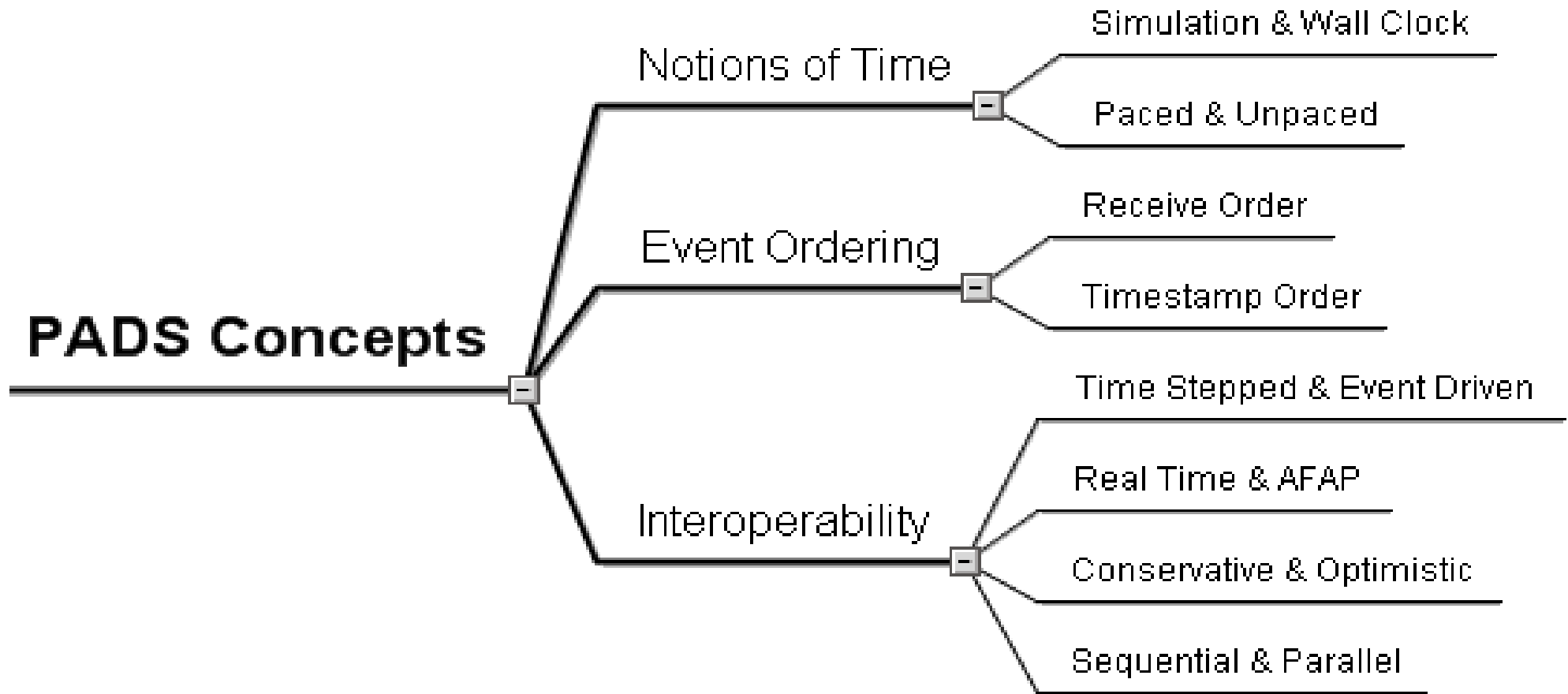
## Terminate execution

- Resign Federation Execution (Federation Mgt)
- Destroy Federation Execution (Federation Mgt)

Rest of the tutorial will focus on Time Management Services in the HLA

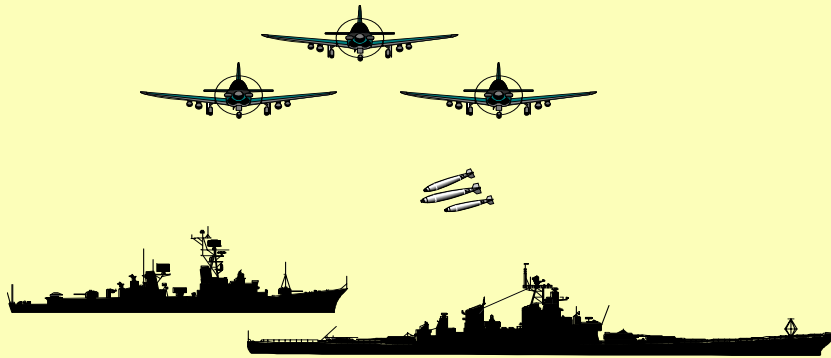
- Parallel and Distributed Simulation (PADS) research previously explored issues in time synchronized simulations
- HLA TM built on insights from PADS research
- Fundamental concepts in HLA TM are common with those in PADS

## 2. PADS Fundamental Concepts



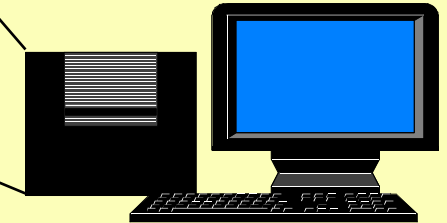
# Notions of Time

- *Physical system*: the actual or imagined system being modeled
- *Simulation*: a system that emulates the behavior of a physical system



*physical system*

```
main()
{ ...
double clock;
...
}
```



*federate (a simulation)*

**Physical time**: time in the physical system (e.g., 0000 to 1700 hours, December 7, 1941)

- Simulation time**: representation of physical time within the simulation
- *federation time axis (FTA)*: a totally ordered sequence of values representing physical time (floating point values in interval [0.0, 17.0])
  - *federate time*: a specific federate's current point on FTA (e.g., 4.0)

**Wallclock time**: time during the execution of the simulation, usually output from a hardware clock (e.g., 1230 to 1400 hours on December 1, 2003)

There may or may not be a specific relationship between simulation time and wallclock time.

### Modes of execution

*As-fast-as-possible* execution (unpaced): no fixed relationship necessarily exists between advances in simulation time and advances in wallclock time

*Real-time* execution (paced): each advance in simulation time is paced to occur in synchrony with an equivalent advance in wallclock time

*Scaled real-time* execution (paced): each advance in simulation time is paced to occur in synchrony with  $S$  \* an equivalent advance in wallclock time (e.g., 2x wallclock time)

# Event Orderings

The HLA provides two types of message ordering:

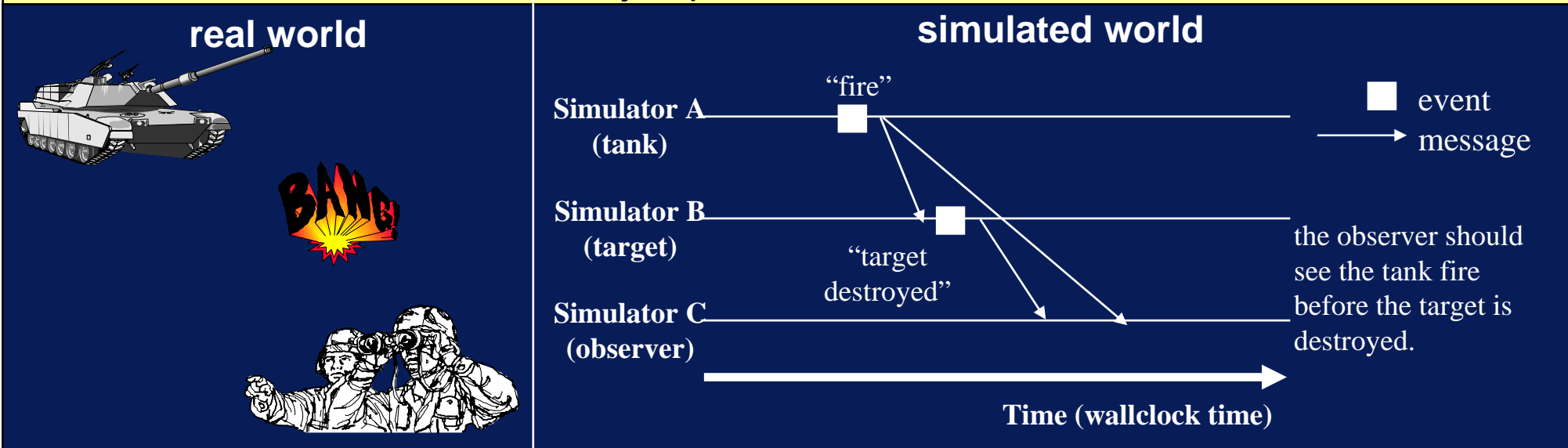
- ***Receive order (unordered)***: messages passed to federate in an arbitrary order
- ***Time stamp order (TSO)***: sender assigns a time stamp to message; successive messages passed to each federate have non-decreasing time stamps

Property	Receive Order (RO)	Time Stamp Order (TSO)
Latency	<i>low</i>	<i>higher</i>
Reproduce before and after relationships?	<i>no</i>	<i>yes</i>
All federates see same ordering of events?	<i>no</i>	<i>yes</i>
Execution repeatable?	<i>no</i>	<i>yes</i>
Typical applications	<i>training, T&amp;E</i>	<i>analysis</i>

- Receive order minimizes latency, does not prevent temporal anomalies
- TSO prevents temporal anomalies, but has somewhat higher latency

# Accurate Modeling via Timestamp-ordered Events

- “Things” happen in the real world in a certain order (e.g., cause & effect).
- It should appear that events in the simulated world happen in the same order as the real world actions that they represent.



The observer could “see” the target is destroyed before the tank fired upon it!  
Temporal anomalies such as this may or may not be acceptable, depending on the federation’s goals

Correct ordering of events can be achieved by assigning a time stamp (simulation [or logical] time) to each event, and ensuring events are delivered in time stamp order

# Timestamp-Ordered Processing Services

The HLA time management services address two fundamental issues in distributed simulation systems:

## Event order

- Simulation events should be processed in the same order that these events occur in the system being modeled

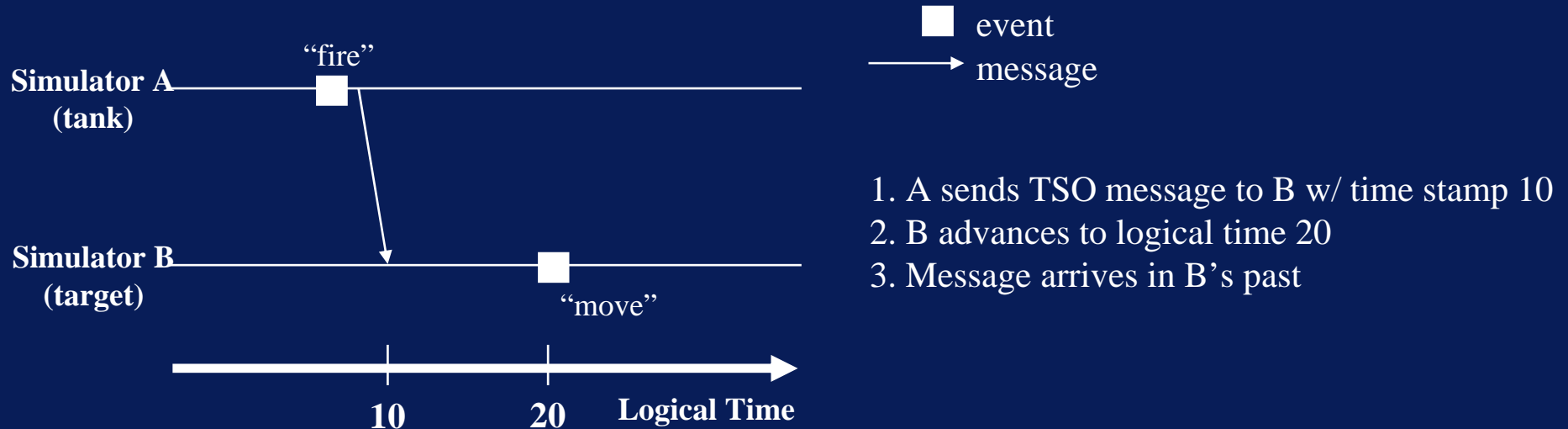
## Time synchronized delivery

- A federate at simulation time  $T$  should not receive events in its past (events with time stamp less than  $T$ )



# Time Synchronized Event Processing

Consider interconnecting two sequential, event driven simulations



Logical time advances by each simulator must be managed to ensure no simulator receives messages in its past.

HLA Time Management (TM) services define a protocol for federates to advance their logical time; RTI will ensure TSO messages are not delivered in a federate's past

# Interoperability Challenge

- Time Stepped
- Event Driven
- Real-Time
- As Fast As Possible
- Sequential
- Parallel
- Conservative
- Optimistic
- ...

constructive federate:  
time-stepped execution

Run Time Infrastructure  
(RTI)

constructive federate:  
event driven execution

Run Time Infrastructure  
(RTI)

training federate:  
real-time execution

Run Time  
Infrastructure (RTI)



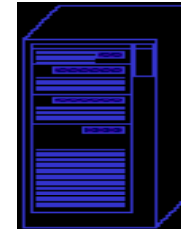
live component:  
real-time execution  
w/ hard deadlines

Run Time  
Infrastructure (RTI)



parallel simulation federate:  
optimistic Time Warp  
execution

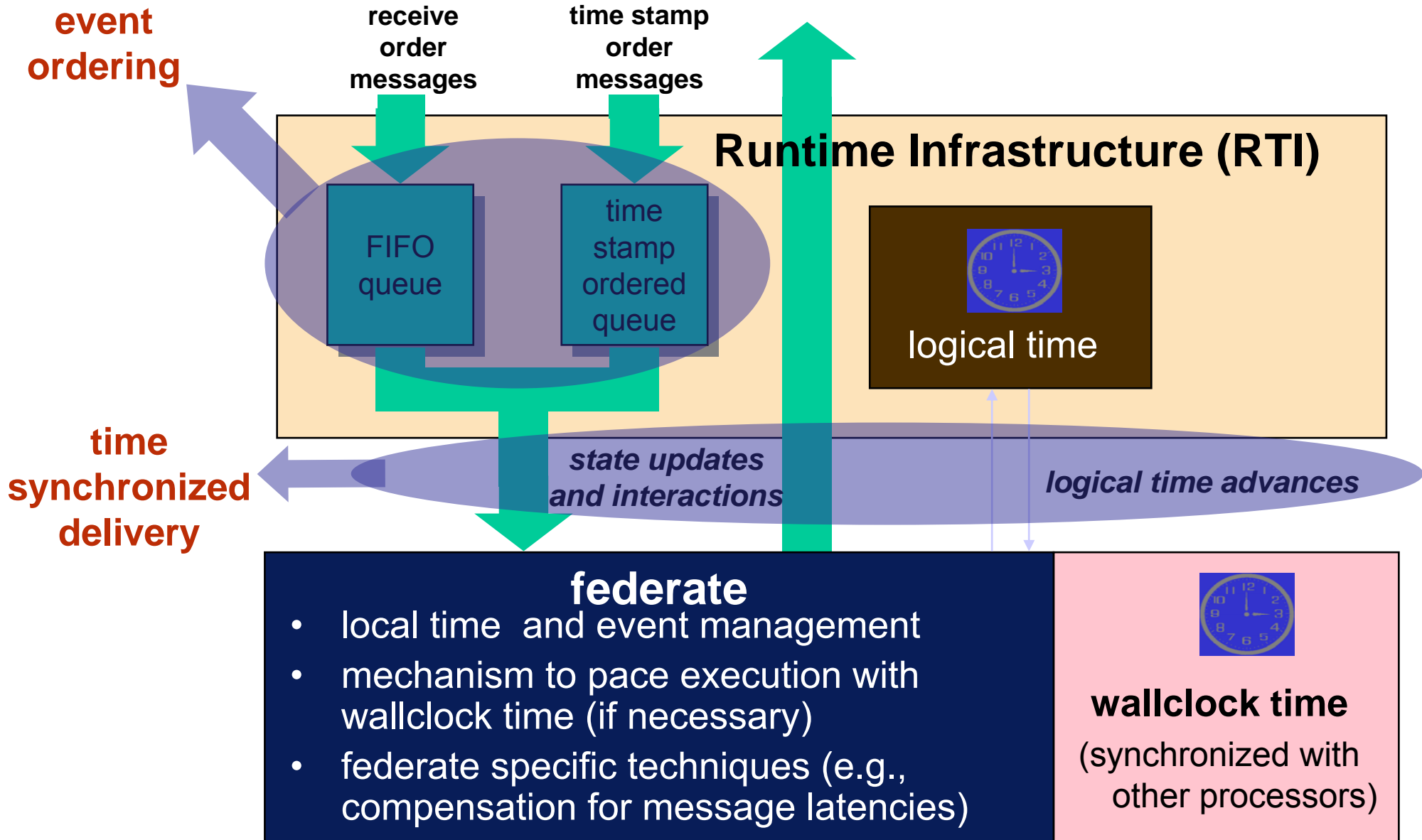
Run Time Infrastructure (RTI)



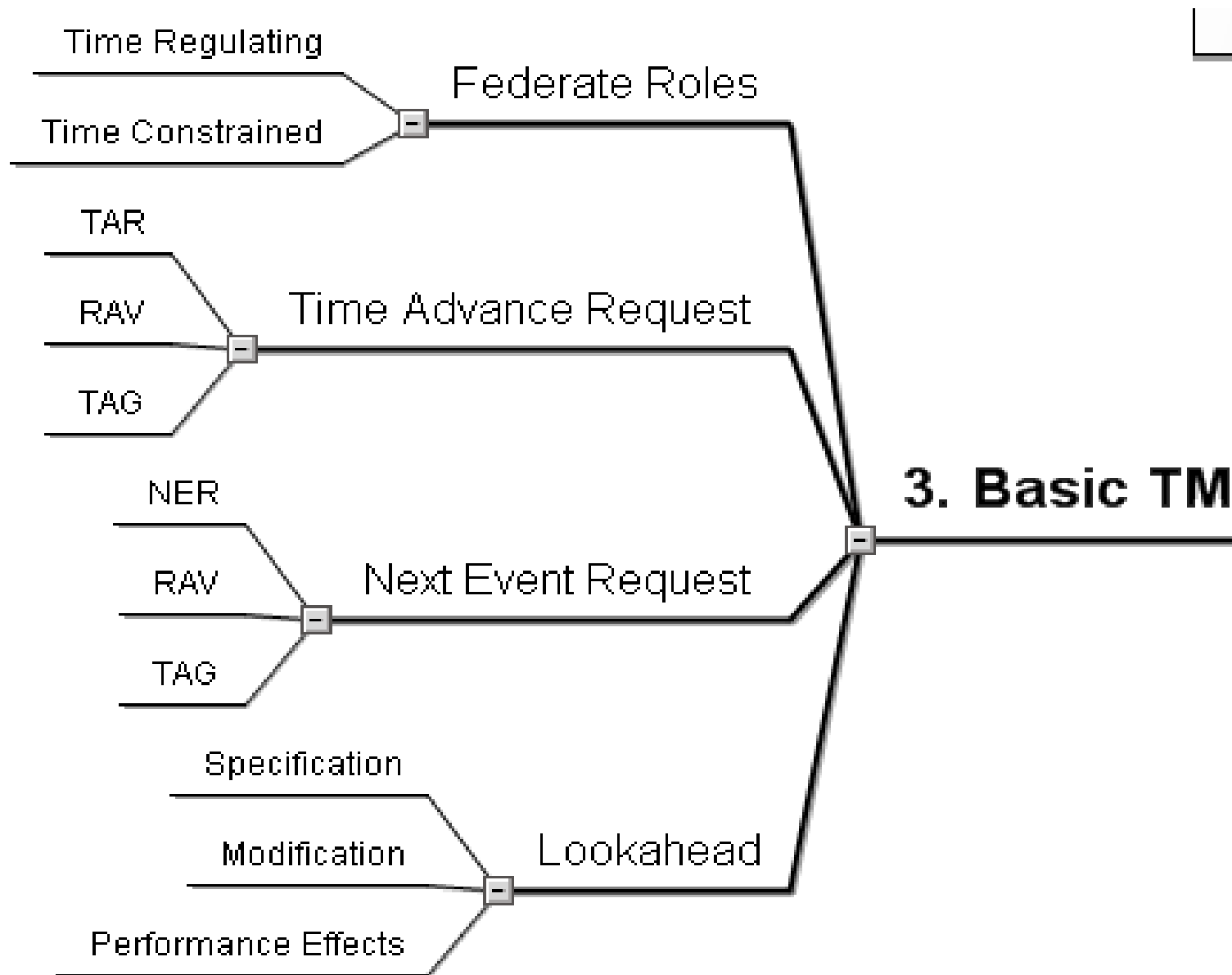
multiprocessor

**Goal:** Provide services to support interoperability among federates with different local time management schemes in a single federation execution

# HLA TM Services



## 2. Basic Time Management



# Federate's Timing Relative to Rest of Federation

Federates must declare their intent to utilize time management services by setting their *time regulating* and/or *time constrained* flags

Time regulating federates: can send TSO messages

- Can prevent other federates from advancing their logical time
- Enable Time Regulation ... **Time Regulation Enabled †**
- Disable Time Regulation

Time constrained federates: can receive TSO messages

- Time advances are constrained by other federates
- Enable Time Constrained ... **Time Constrained Enabled †**
- Disable Time Constrained

Each federate in a federation execution can be

- Time regulating only (e.g., message source)
- Time constrained only (e.g., Stealth)
- Both time constrained and regulating (common case for analytic simulations)
- Neither time constrained nor regulating (e.g., training simulations)

† indicates callback to federate

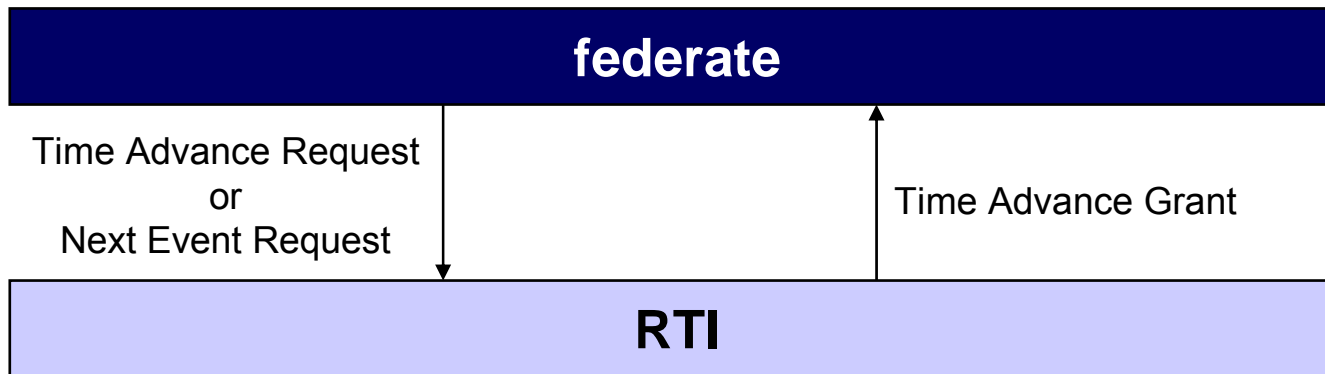
# HLA TM – Time Advancement Protocol

HLA TM services define a protocol for federates to advance logical time; logical time only advances when that federate explicitly requests an advance

**Time Advance Request:** *time stepped federates*

**Next Event Request:** *event driven federates*

**Time Advance Grant:** RTI invokes to acknowledge logical time advances

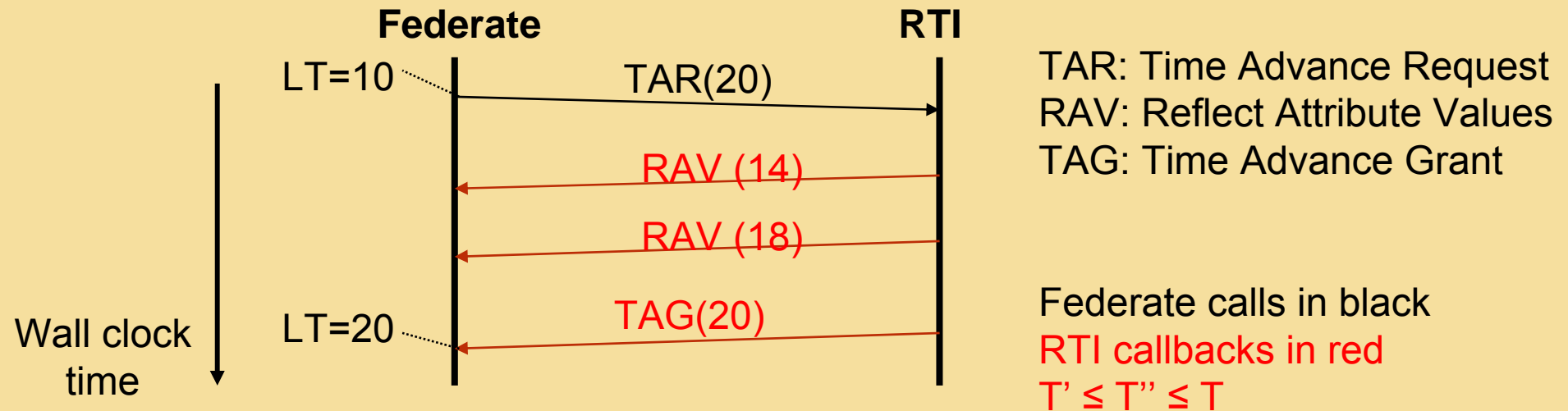


- If the logical time of a federate is  $T$ , the RTI guarantees no more TSO messages will be passed to the federate with time stamp  $< T$
- Federates responsible for pacing logical time advances with wallclock time in real-time executions

# Time Advance Request (TAR)

- Typically used by time stepped federates
- Federate invokes **Time Advance Request (T)** to request its logical time (LT) be advanced to T
- RTI delivers all TSO messages with time stamp  $\leq T$
- RTI advances federate's time to T, invokes **Time Advance Grant (T)** when it can guarantee all TSO messages with time stamp  $\leq T$  have been delivered
- Grant time always matches the requested time

*Typical execution sequence*



# TAR Example – Time Stepped Federate

## Sequential simulator

T = current simulation time

While (simulation not complete)

    update local simulation state

    T = T +  $\Delta T$ ;

End-While

## Federated simulator

While (simulation not complete)

    update local simulation state

    UpdateAttributeValues (...)

    PendingTAR = TRUE;

    TimeAdvanceRequest(T+  $\Delta T$ )

    while (PendingTAR) Tick\*(...);

    T = T +  $\Delta T$ ;

End-While

*/\* the following federate-defined  
procedures are called by the RTI \*/*

Procedure ReflectAttributeValues (...)

    update local state

Procedure TimeAdvanceGrant (...)

    PendingTAR = False;

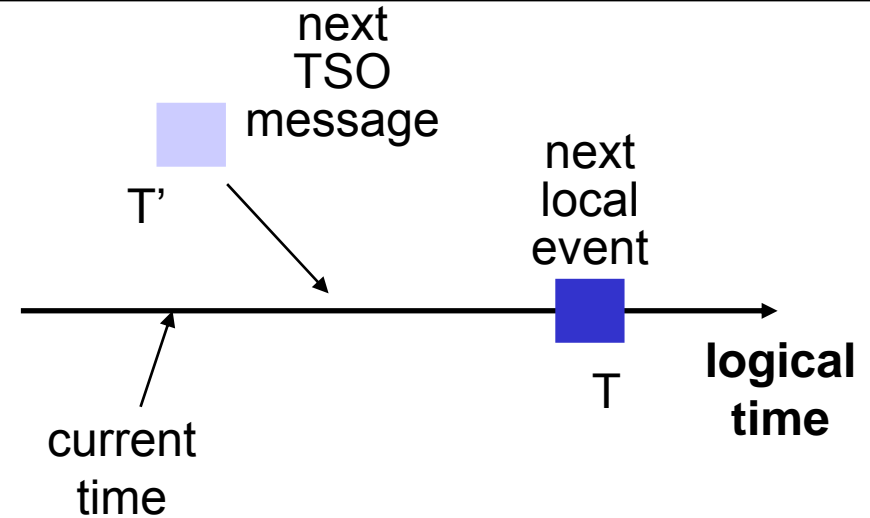
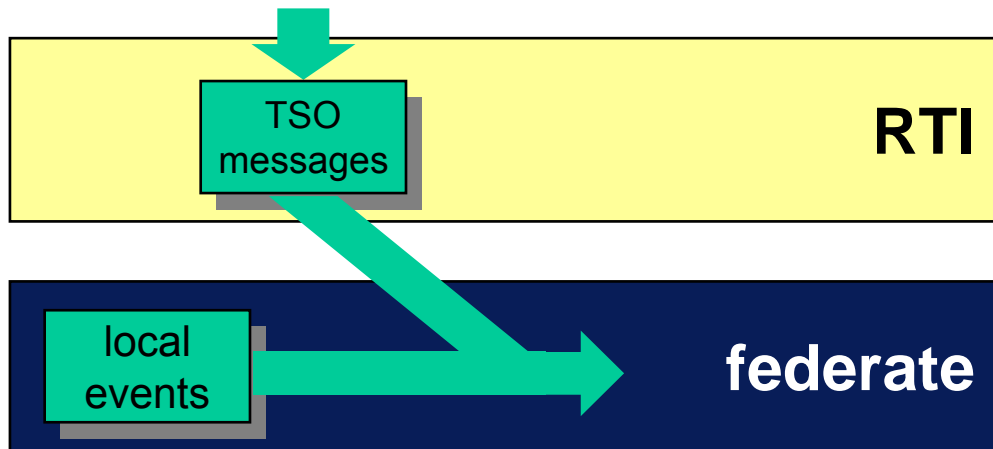
\* Tick is only used in single threaded RTI implementations



# Next Event Request (NER)

Typically used by event stepped federates

Goal: process all events (local and incoming TSO messages) in time stamp order



Federate's View: Next local event has time stamp  $T$ ; Two cases:

Case 1: If no TSO messages w/ time stamp  $< T$  are forthcoming, advance to  $T$ , process local event

Case 2: If there is a TSO message w/ time stamp  $T' \leq T$ , advance to  $T'$  and process TSO message

## NER (Continued)

Federate invokes **Next Event Request (T)** to request its logical time be advanced to time stamp of next TSO message, or T, whichever is smaller

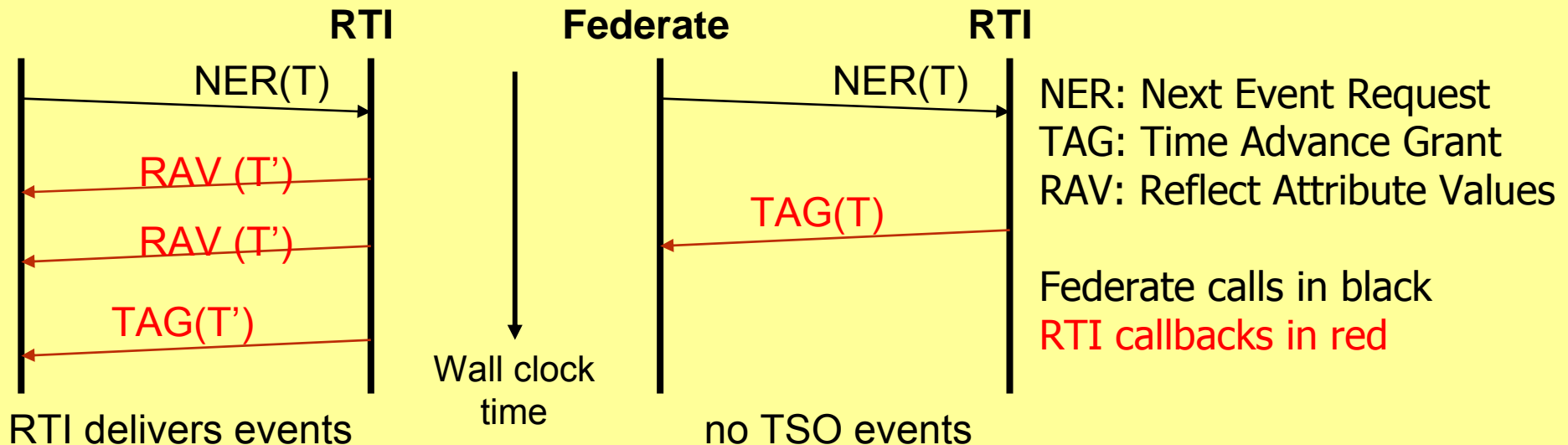
If next TSO message has time stamp  $T' \leq T$

- RTI delivers next TSO message, and all others with time stamp  $T'$
- RTI issues **Time Advance Grant (T')**

Else

- RTI advances federate's time to T, invokes **Time Advance Grant (T)**

### Typical execution sequences



# NER Example – Event-Driven Federate

## Sequential simulator

T = current simulation time

PES = pending event set

While (simulation not complete)

    T = time of next event in PES

    process next event in PES

End-While

## Federated simulator

While (simulation not complete)

    T = time of next event in PES

    PendingNER = TRUE;

    NextEventRequest(T)

    while (PendingNER) Tick(...);

    process next event in PES

End-While

*/\* the following federate-defined  
procedures are called by the RTI \*/*

Procedure ReflectAttributeValues (...)

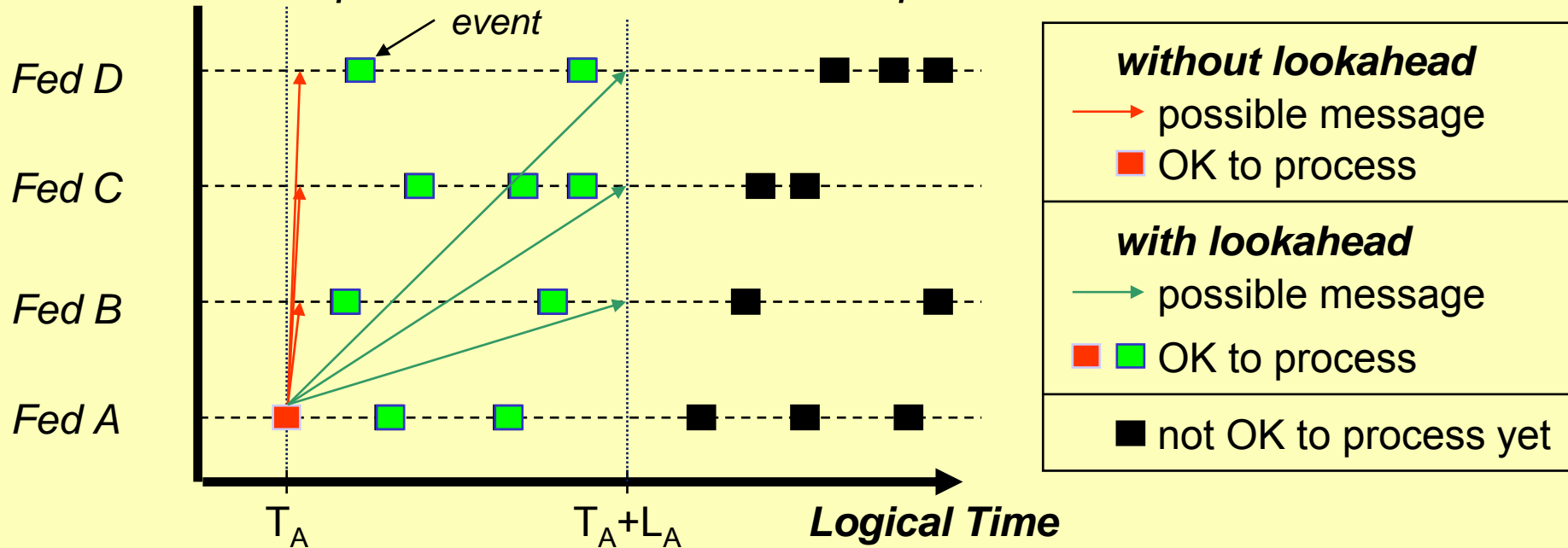
    place event in PES

Procedure TimeAdvanceGrant (...)

    PendingNER = False;

# Lookahead

Problem: Limited concurrency in event driven federates  
Each federate must process events in time stamp order



Fed A declares a lookahead value  $L_A$ ; the time stamp of any event generated by the Fed A must be  $\geq T_A + L_A$

- Used in virtually all conservative synchronization protocols
- Relies on model properties (e.g., minimum interaction delay)

Lookahead is necessary to allow concurrent processing of events with different time stamps (unless optimistic event processing is used)

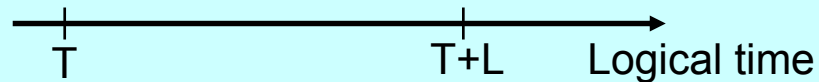
# Lookahead

Each federate must declare a non-negative lookahead value

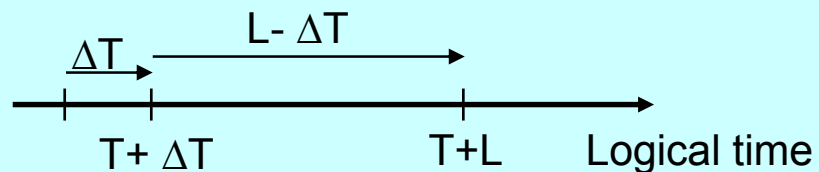
Any TSO sent by a federate must have time stamp at least the federate's current time plus its lookahead

Lookahead can change during the execution (*Modify Lookahead*)

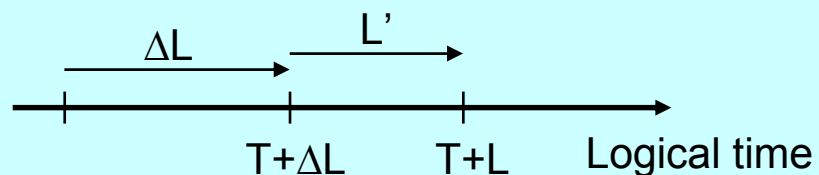
- increases take effect immediately
- decreased do not take effect until the federate advances its logical time



1. Current time is  $T$ , lookahead  $L$
2. Request lookahead decrease by  $\Delta L$  to  $L'$

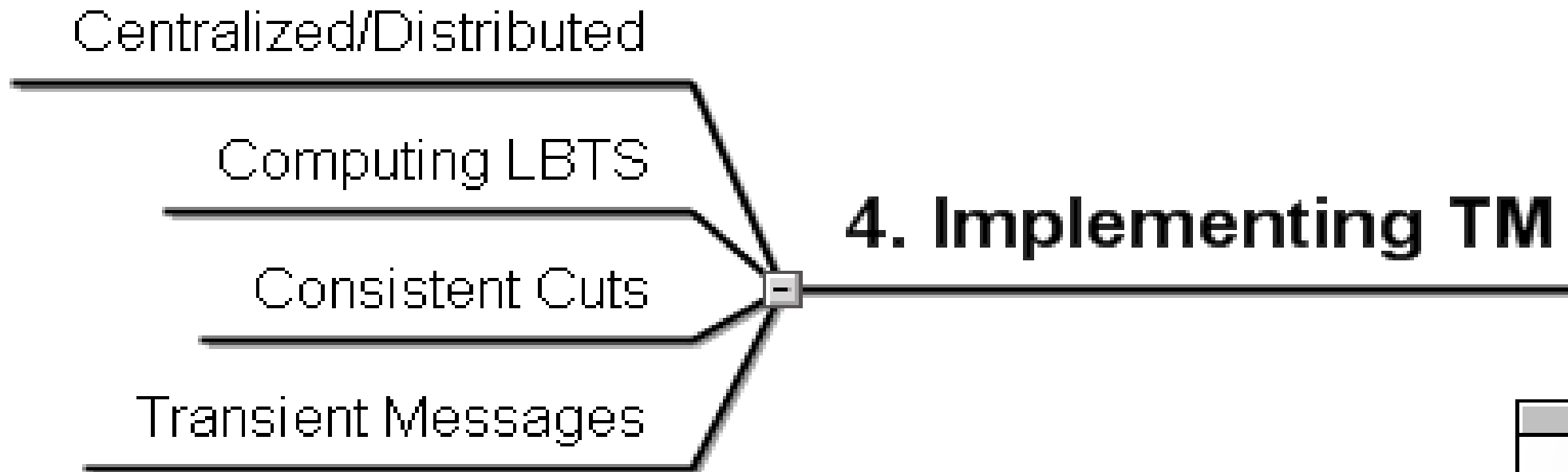


3. Advance  $\Delta T$ , lookahead, decreases  $\Delta T$



4. After advancing  $\Delta L$ , lookahead is  $L'$

## 4. Implementing TM – Inside the RTI



# Terminology Difference: IEEE 1516 vs. RTI 1.3NG

## IEEE 1516

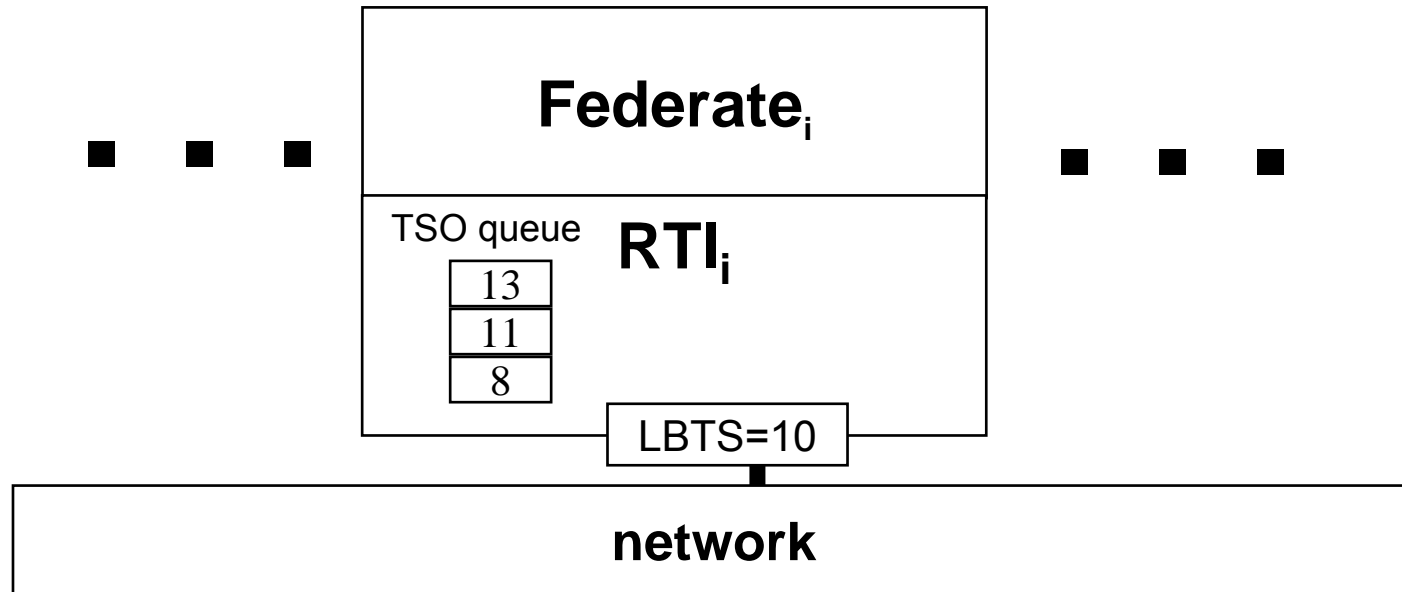
- GALT: Greatest Available Logical Time
- Denotes the time to which a federate can safely advance its logical time
- No incoming (externally generated) events will have timestamps less than GALT

## RTI 1.3NG

- LBTS: Lower Bound on Time Stamp
- Denotes the earliest time stamp of any event that a federate can receive from now to anytime in future
- Implies the federate can safely advance its local logical time to LBTS
  
- We will use LBTS in rest of the tutorial
- Can safely substitute LBTS with GALT

# TM Implementation Requirements

- Deliver messages to federate in time stamp order
- Ensure federate does not receive an event in its past



**LBTS<sub>i</sub>**: **L**ower **B**ound on **T**ime **S**tamp of TSO messages that could later be placed into the TSO queue for federate  $i$

TSO messages w/  $TS \leq LBTS_i$  eligible for delivery

RTI ensures logical time of federate  $i$  never exceeds  $\min(LBTS_i, TSO_{\min-i})$  where  $TSO_{\min-i}$  = minimum time stamp of messages in local TSO queue



# Centralized vs. Distributed

TM services can be implemented in two ways:

- Centralized approach
- Fully distributed approach

## Centralized Approach:

- A single computer acts as RTI's "time management gateway"
- All time-synchronized operations are routed through this gateway
  - Time advance requests, grants, etc. are coordinated by gateway
- Advantages: Simplicity of design; easier to debug and test
- Disadvantages: Single point of failure; potentially high runtime overhead

## Distributed Approach:

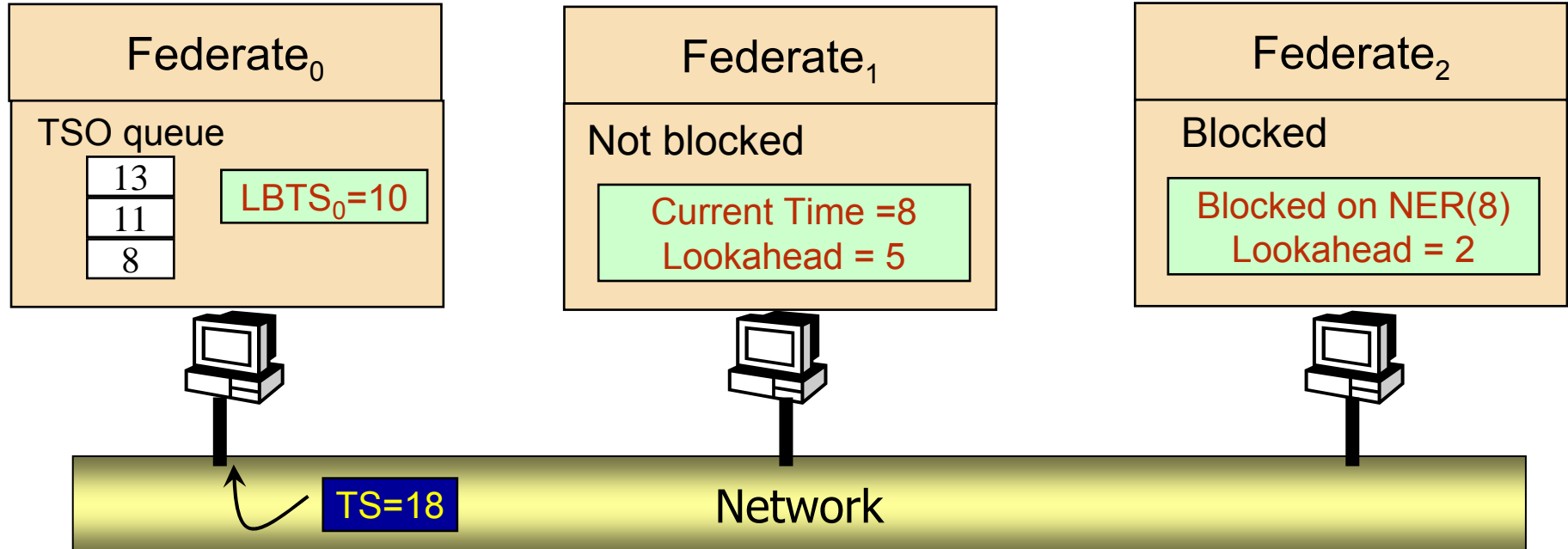
- All federates' nodes *directly* undertake synchronization roles
- Local time-synchronized
- Advantages: Faster execution; more uniform synchronization load
- Disadvantages: Greater implementation complexity

Here, we will focus on the distributed approach

# Computing LBTS

Given a snapshot of the computation, LBTS is minimum among

- Time stamp of any transient messages (sent, but not yet received)
- Unblocked LPs: Current simulation time + lookahead
- Blocked LPs:
  - TAR: Time of parameter in TAR call + lookahead
  - NER: Time of parameter in NER call + lookahead

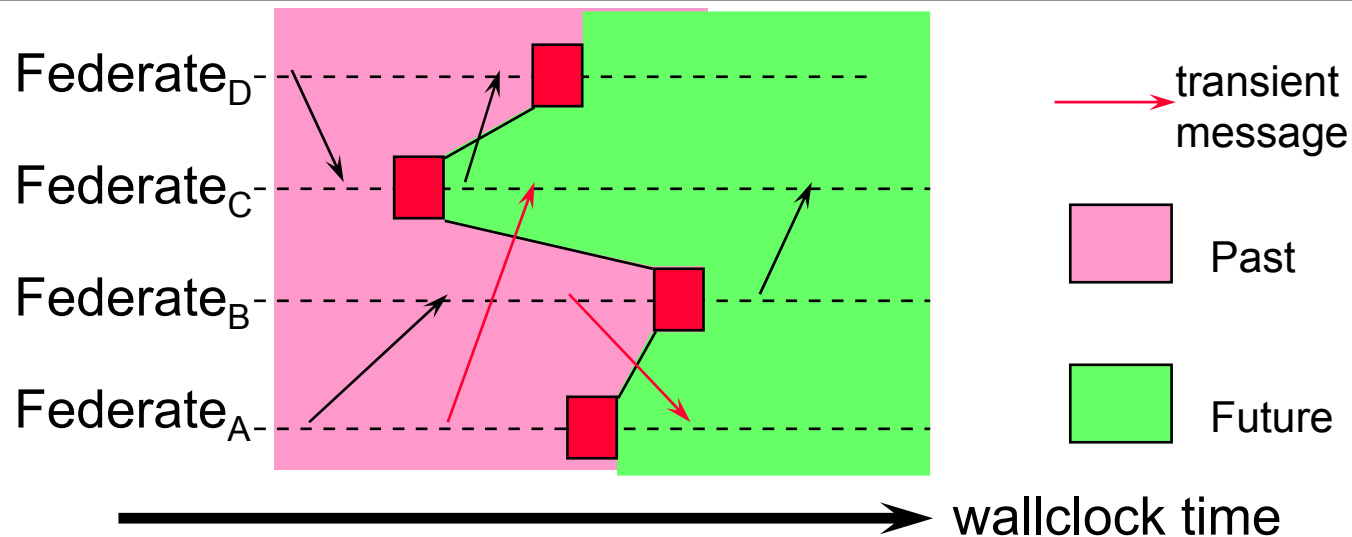


# Consistent Cuts

**cut point:** an instant dividing computation into past and future

**cut:** set of cut points, one per processor

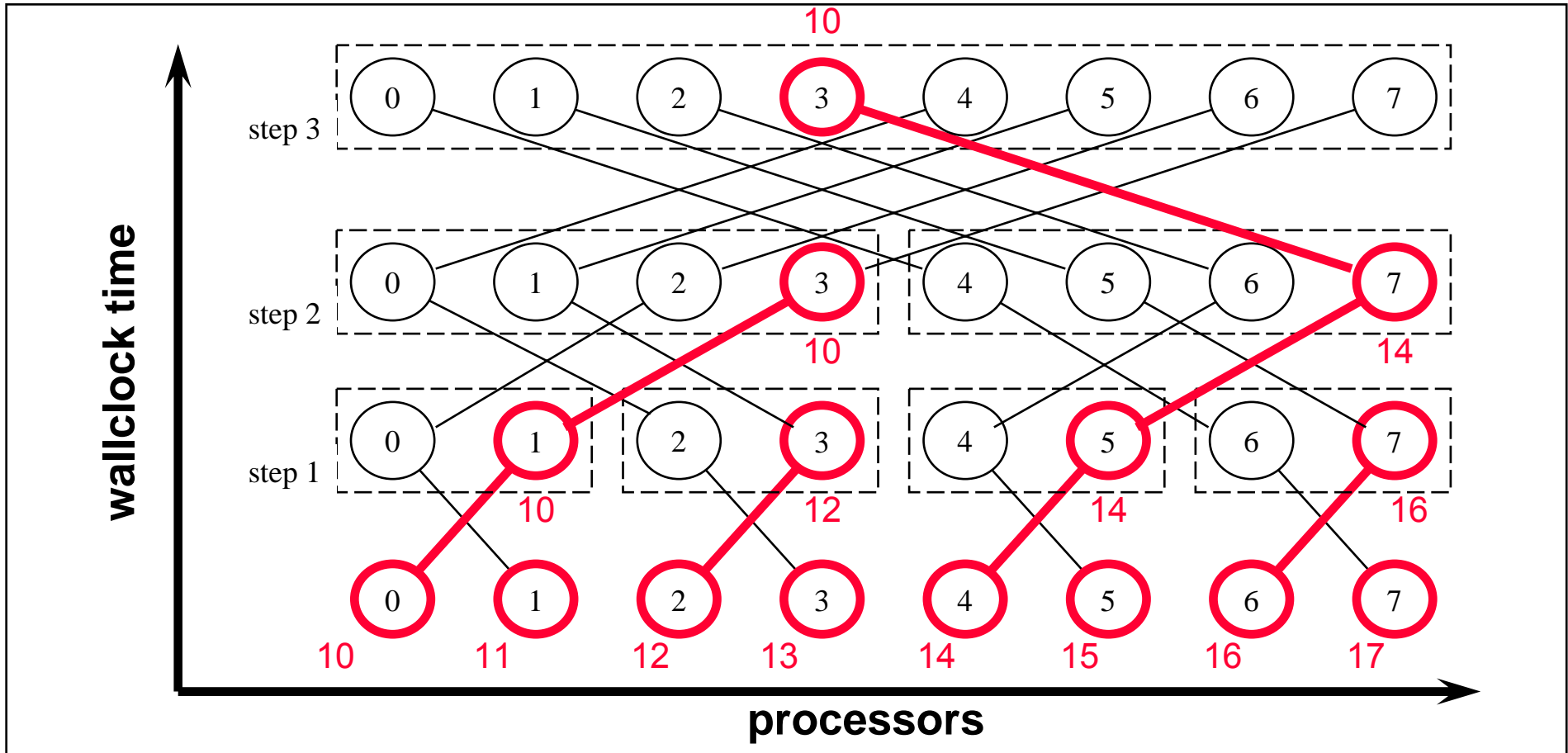
**consistent cut:** a cut where all messages crossing the cut were sent in the past and received in the future [Mattern]



LBTS = minimum time value along a consistent cut:

- a local minimum of each processor at its cut point
- time stamp of *transient messages* (cross cut past to future)
- LBTS algorithm: (1) global minimum (2) transient messages

# Computing Global Minimum – Distributed Reduction



Step  $i$ , processor  $k$  executes ( $\log N$  steps for  $N$  processors):

- Pairwise minimum with processor  $j$ ,  $j = (k w/ \text{bit position } i \text{ inverted})$
- After  $\log N$  steps each processor has a copy of the global minimum
- Can generalize to any number of processors, configurable fan-out

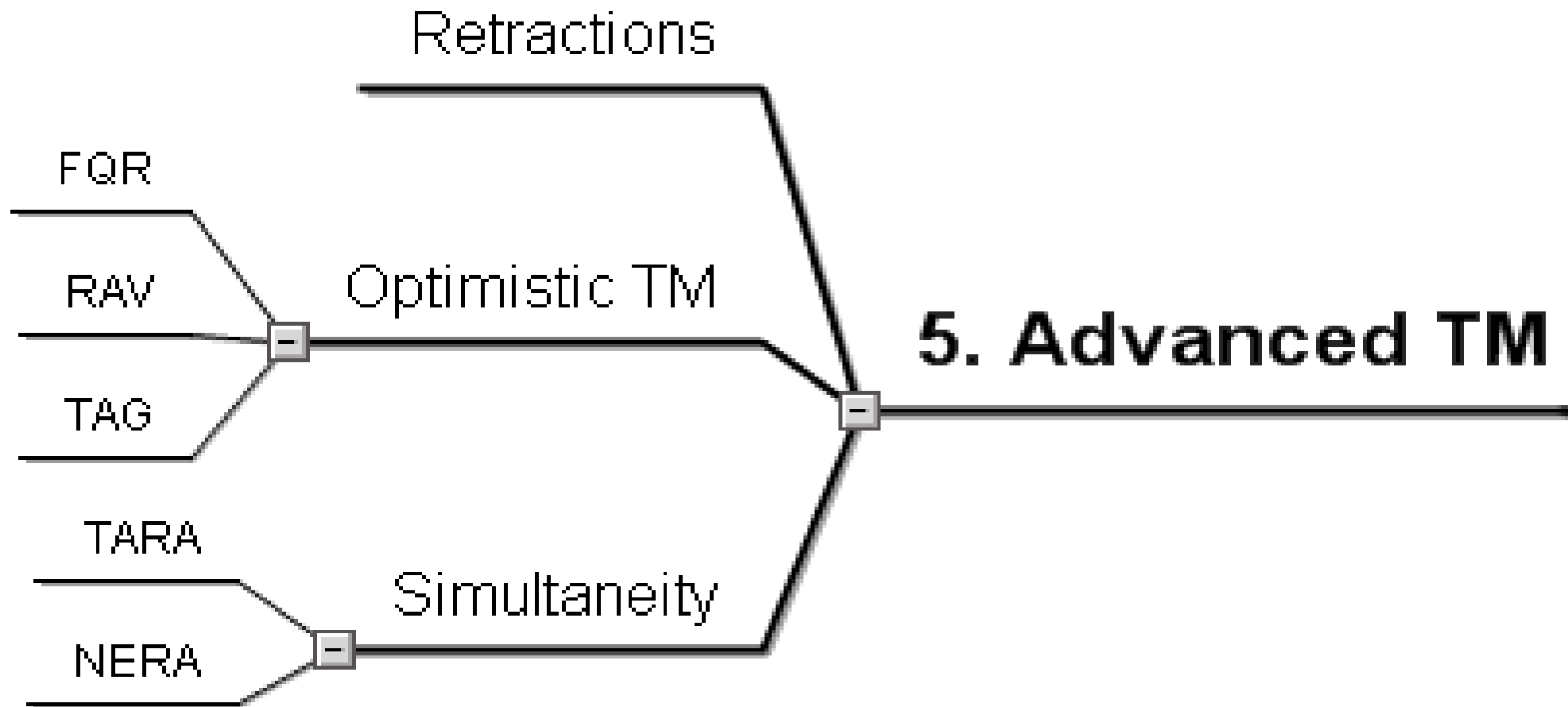
# Transient Messages

- Basic idea: check there are no transient messages (all messages sent prior to the cut have been received)
- Color messages
  - federate colored red before cut point, green after cut point
  - message color is the color of the federate when it sent the message
  - transient message: red message received by green federate
- Detect receipt of all transient message
  - $S_i$  = number of red messages sent by federate  $i$
  - $R_i$  = number of red messages received by federate  $i$
  - when  $\sum S_i = \sum R_i$ , all red messages have been received (no transients)

## One implementation

- Maintains one counter ( $S_i - R_i$ ) for each federate  $i$
- Piggyback summation of counters with global reduction computation
- If counters sum to zero, done; else, retry

## 5. Advanced Time Management

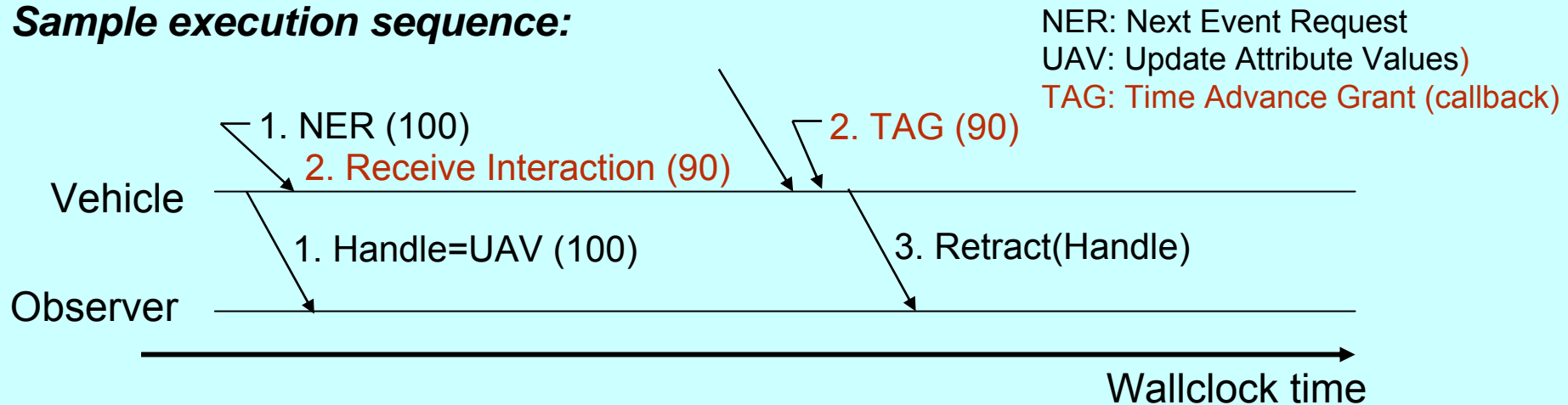


# Retractions

Previously sent events can be “unsent” via the *Retract* service

- Update Attribute Values and Send Interaction return a “handle” to the scheduled event
- Handle can be used to Retract (unschedule) the event
- Can only retract event if its time stamp > current time + lookahead
- Retracted event never delivered to destination (unless *Flush Queue* used)

## Sample execution sequence:



1. Vehicle schedules position update at time 100, ready to advance to time 100
2. receives interaction (break down event) invalidating position update at time 100
3. Vehicle retracts update scheduled for time 100

Time Management algorithms are often categorized as conservative or optimistic

- Conservative

- Avoid synchronization errors (message in federate's past)
- Use blocking to avoid errors

- Optimistic

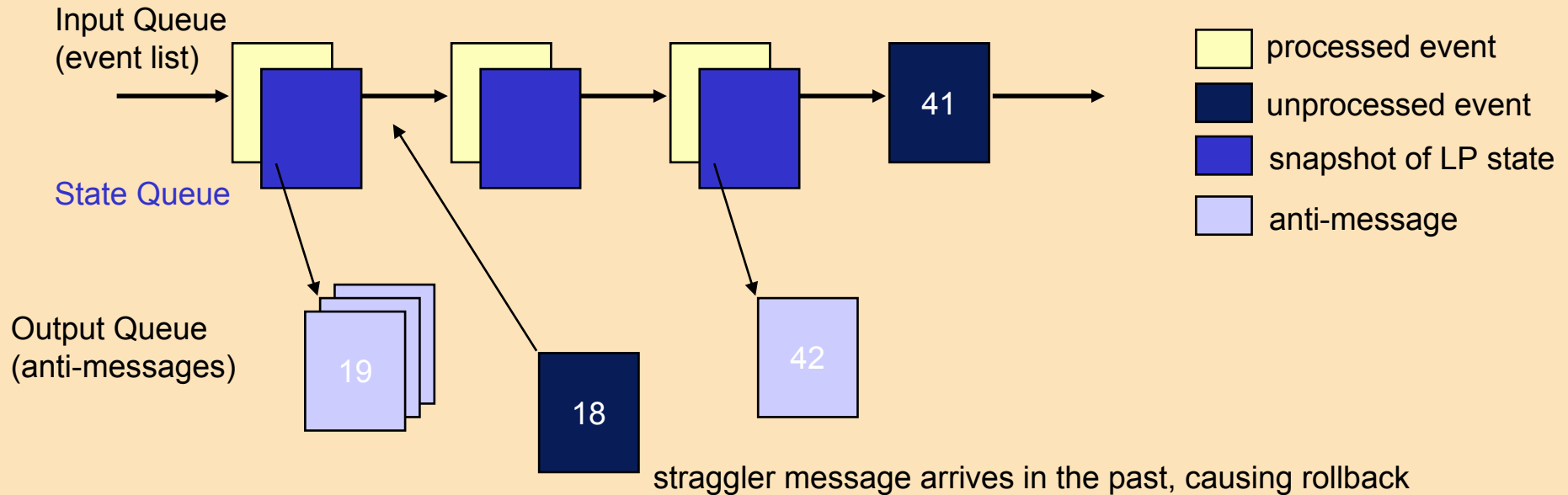
- Do not block; process messages without worrying about messages that might arrive later
- Detect synchronization errors during the execution
- Recover using a rollback mechanism

- Optimistic processing offers better concurrency, less reliance on lookahead



# Time Warp

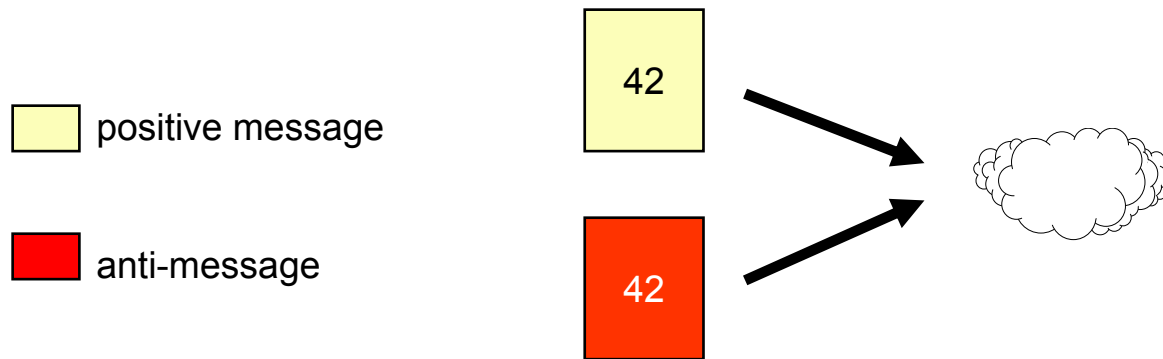
Federate: process events in time stamp order, like a sequential simulator, except: (1) do NOT discard processed events and (2) add a rollback mechanism



Adding rollback:

- a message arriving in the federate's past initiates rollback
- to roll back an event computation we must undo:
  - changes to state variables performed by the event;  
*solution: checkpoint state or use incremental state saving (state queue)*
  - message sends  
*solution: anti-messages and message annihilation (output queue)*

# Anti-Messages In Optimistic Federates



Used to cancel a previously sent message

Each positive message sent by a federate has a corresponding anti-message

Anti-message is identical to positive message, except for a sign bit

When an anti-message and its matching positive message meet in the same queue, the two annihilate each other (analogous to matter and anti-matter)

To undo the effects of a previously sent (positive) message, the federate need only send the corresponding anti-message

Message send: in addition to sending the message, leave a copy of the corresponding anti-message in a data structure in the sending federate called the output queue.

# HLA Support for Optimistic TM

## HLA Federations

- federations may include conservative and/or optimistic federates
- federates not aware of local time management mechanism of other federates (optimistic or conservative)
- optimistic events (events that may be later canceled) will not be delivered to conservative federates that cannot roll back
- optimistic events can be delivered to other optimistic federates

## • Optimistic event processing

- Deliver (and process) events without time stamp order delivery guarantee
- HLA: **Flush Queue Request**

## • Rollback

- HLA: (local) rollback mechanism must be implemented within the federate

## • Anti-messages & secondary rollbacks

- Anti-message: message sent to cancel (undo) a previously sent message
- HLA: **Retract** service; deliver retract request if cancelled message already delivered

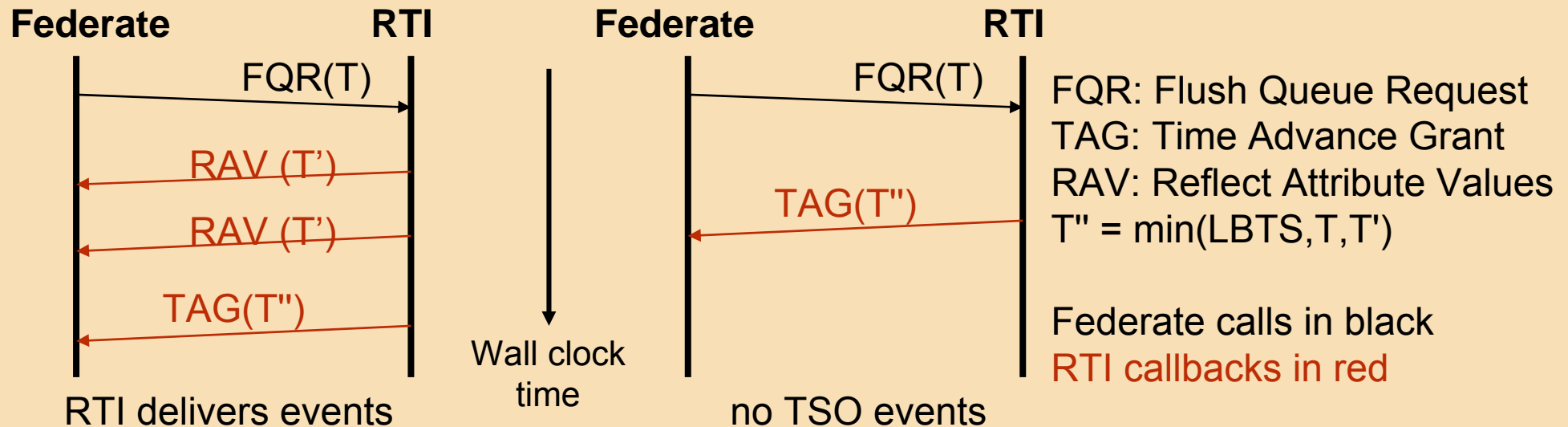
## • Federate time used to compute lower bound on time stamp of future rollbacks (Global Virtual Time)

# Flush Queue Request

**Flush Queue Request:** similar to NER except

- (1) deliver all messages in RTI's local message queues,
- (2) need not wait for other federates before issuing a Time Advance Grant

*Typical execution sequences*



# Simultaneity of Events and Zero Lookahead

**Simultaneous Events:** Events containing the same timestamps

- Ordering of simultaneous events often important
- RTI does not have sufficient information to intelligently order simultaneous events

**HLA Policy:** Ordering simultaneous events left to the federate

- Grant to time T (after TAR/NER): all events with time stamp T delivered to federate
- Simultaneous events delivered to federate in an arbitrary order (may vary from one execution to the next)

**A Source of Simultaneity:** Equal timestamps can be generated due to chain of events sent, each sent with zero lookahead

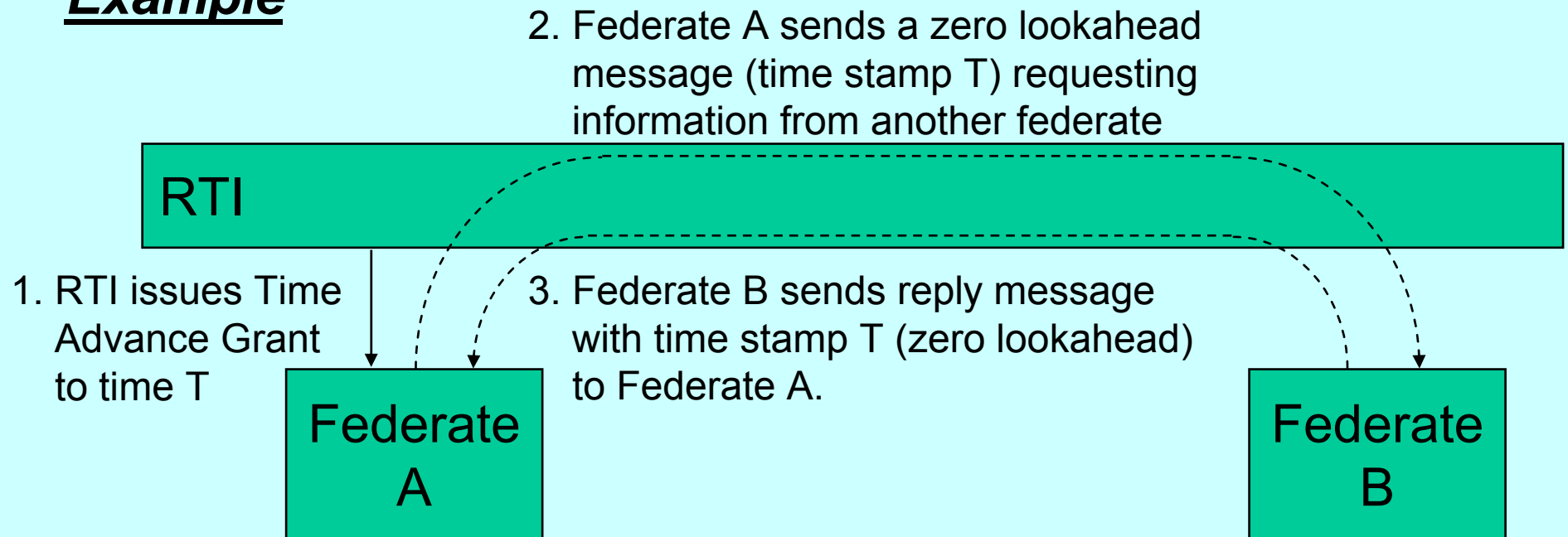
**Problem:** Zero lookahead incompatible with NER & TAR definitions...

# Zero Lookahead

**Zero Lookahead:** A federate at time  $T$  can send TSO events with timestamp  $T$ .

With zero lookahead, a Time Advance Grant to time  $T$  cannot guarantee delivery of all events with time stamp equal to  $T$

## Example



**HLA Solution:** Two new services: NERA & TARA

## Next Event Request Available (NERA)

- Federate can send zero lookahead messages after receiving grant
- Grant to time T does *not* guarantee all messages with time stamp T have been delivered, only those available at the time of the call
- Order that TSO messages are delivered to the federate is arbitrary

## Time Advance Request Available (TARA)

- TARA is the zero-lookahead counterpart for TAR
- Analogous to NERA for NER

# Example – NERA and Zero Lookahead

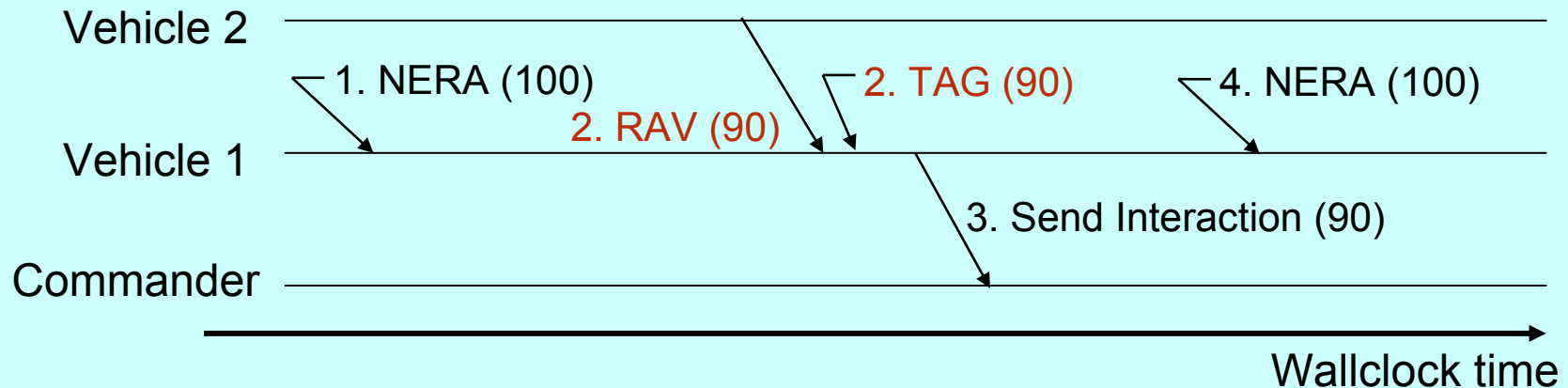
Two federate types: vehicle, commander

Vehicle federate

- When sensors indicate position of another vehicle has changed, notify commander via a zero lookahead interaction

**Sample execution sequence:**

NERA: Next Event Request Available  
RAV: Reflect Attribute Values (callback)  
TAG: Time Advance Grant (callback)



1. Vehicle 1 ready to advance to logical time 100 to process next local event
2. Detects vehicle 2 has moved
3. Sends zero lookahead interaction to commander federate
4. Ready again to advance to 100; additional reflects at time 90 still possible



# Summary

## **HLA Time Management Functionality:**

Allows federates with different local time management mechanisms to be combined within a single federation execution

- Event-driven or Time-stepped simulations, Optimistic parallel simulations, ...

## **HLA Time Management services:**

- Event order
  - Receive order delivery (lesser runtime overhead)
  - Timestamp order delivery (better modeling accuracy)
- Logical time advance mechanisms
  - Time Advance Request: time stepped federates
  - Next Event Request: event driven federates
  - Flush Queue Request: optimistic federates
  - Variants for zero lookahead processing
- Lookahead to increase concurrency

## **Implementation of HLA Time Management services:**

- Compute LBTS on future messages a federate might later receive
- Several algorithms exist, e.g., distributed snapshots

# References

## Books

Richard Fujimoto, "Parallel and Distributed Simulation Systems," Wiley Interscience, ISBN 0471183830 (2000).

Frederick Kuhl, Richard Weatherly and Judith Dahmann, "Creating Computer Simulation Systems: An Introduction to the High Level Architecture," ISBN 0130225118 (1999).

## Overview Papers (see [www.ornl.gov/~2ip](http://www.ornl.gov/~2ip) )

Kalyan Perumalla, "Parallel and Distributed Simulation Systems and the High Level Architecture," I/ITSEC (2005).

Kalyan Perumalla, "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances," Winter Simulation Conference (2006).

# Acknowledgements

Majority of the slides here are based on the tutorial in 2003 by Dr. Richard Fujimoto (Georgia Tech). Dr. Fujimoto was a lead architect in the design of the HLA TM.

Guidance, critique and helpful suggestions for improvement from Andreas Tolk (Old Dominion University) are gratefully acknowledged. Andreas Tolk has been the “bird dog” for this tutorial.