

Tutorial

# Parallel and Distributed Simulation (PADS): Traditional Techniques & Recent Advances

Kalyan Perumalla, Ph.D.  
Senior Researcher

Oak Ridge National Laboratory

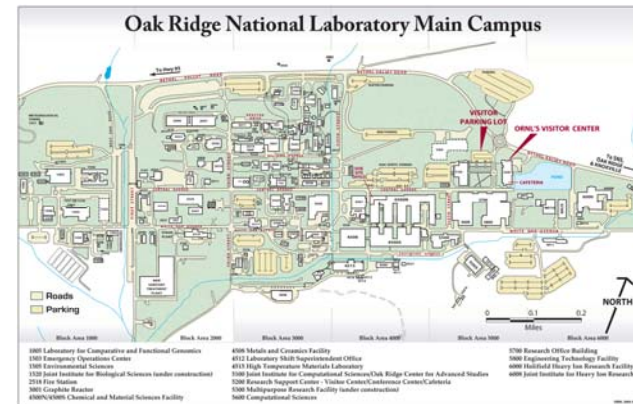
[perumallaks@ornl.gov](mailto:perumallaks@ornl.gov)

[www.ornl.gov/~2ip](http://www.ornl.gov/~2ip)

June 12, 2007

# Where is ORNL? Where is Oak Ridge?

## Tennessee



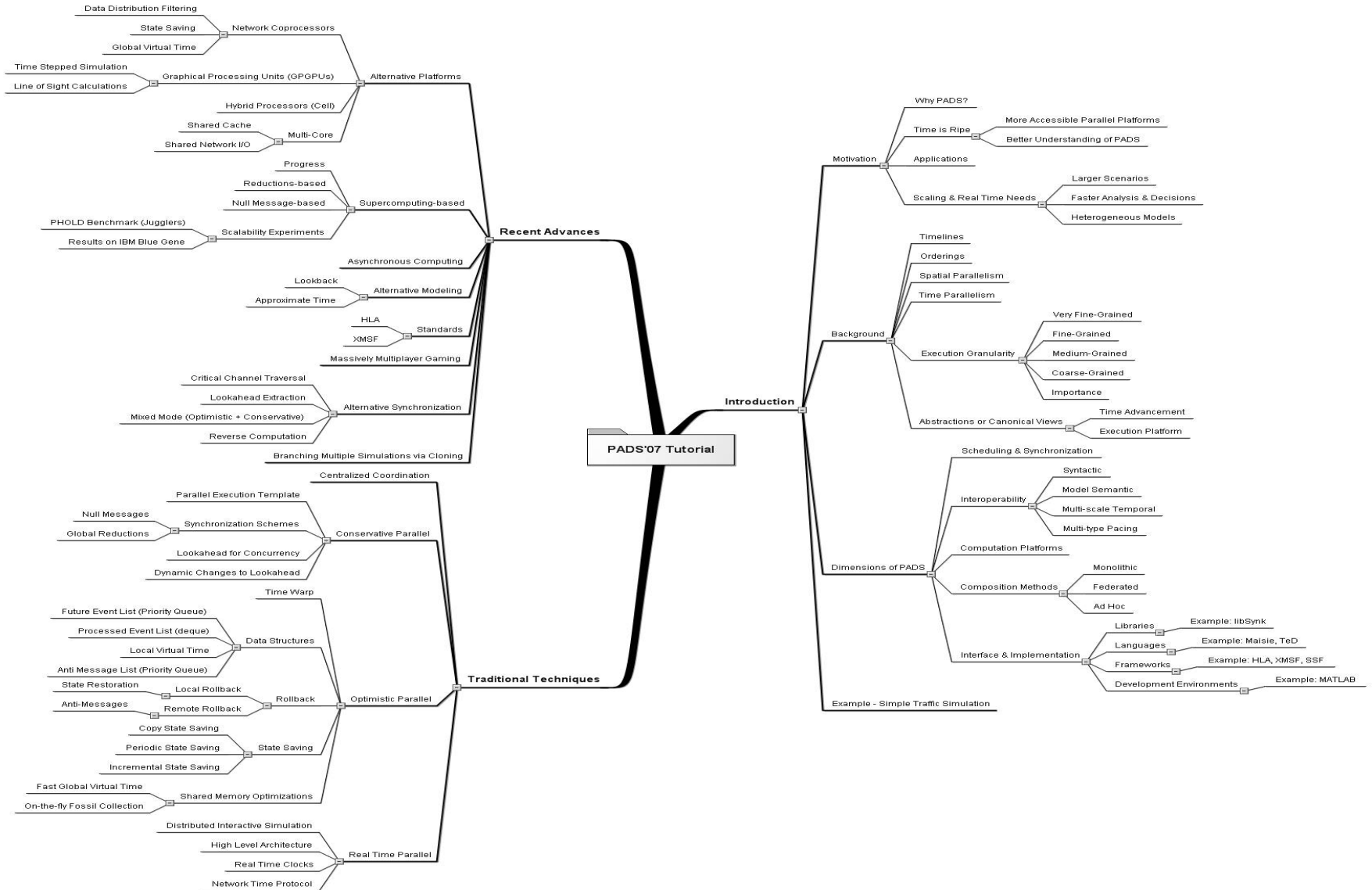
## Oak Ridge



2.5 hour drive

- Introduction
- Traditional Techniques
- Recent Advances

# Outline



- Motivation
- Background
- Example Simulation

## What is a “Parallel and Distributed Simulation”?

Any simulation in which more than one processor is employed.

Other definitions are possible, but we will use the above one.

- Why use PADS?
- What is its Relevance Now?
- Applications
- Scale & Real Time Needs

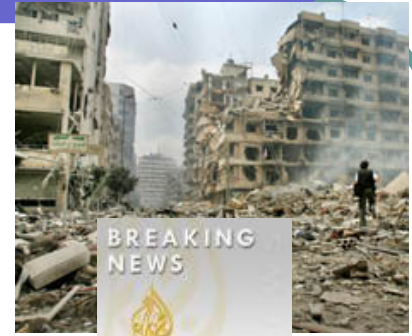
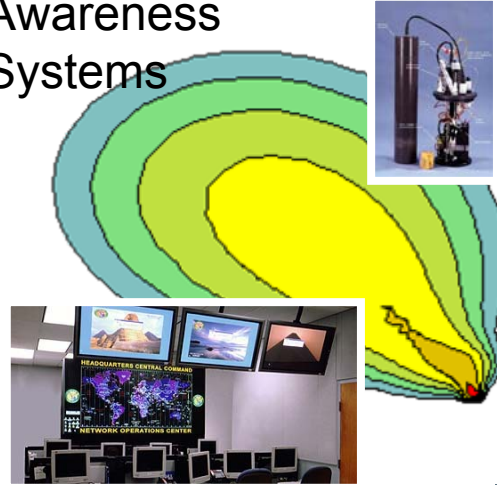
# Why use PADS?

- Complete the simulation faster
  - with larger no. of processors
- Simulate larger scenarios
  - using greater amount of memory & resources
- Integrate “inherently separated” simulators
  - e.g., geographically distributed
- Integrate proprietary simulators
  - e.g., commercial off the shelf tools
- Realize enhanced functionality
  - e.g., composing multiple disparate models

# Examples of PADS Application Domains

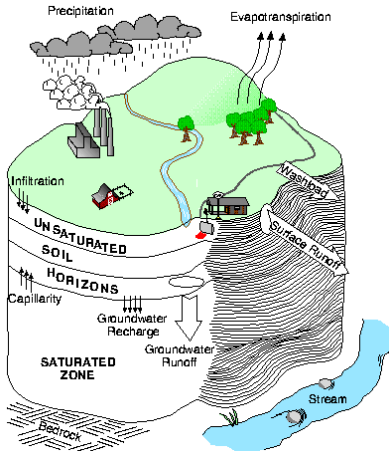


## Protection & Awareness Systems



## Global & Local Events

## NBC Incidents & Effects



## Current & Future Defense Systems



## Critical Infrastructures

# Selected PDES Applications

- Network simulation
  - Internet protocols, Security , P2P designs, ...
- Traffic simulation
  - Emergency planning/response, Environmental policy analysis, Urban planning, ...
- Social dynamics simulation
  - Operations planning, Foreign policy, Marketing, ...
- Sensor simulations
  - Wide area monitoring, Situational awareness, Border surveillance, ...
- Organization simulations
  - Command & control, Business processes, ...

# Scaling & Real Time Needs

- **Larger Scenarios**

Examples:

- Larger-sized grids in scientific computing
- Greater counts of entities in agent models
- Large number of nodes in computer networks

- **Faster Analysis & Decisions**

Examples:

- A few hours per emergency decision-making
- Monte Carlo exploration of large-scale phenomena using parallel simulation per scenario run

- **Heterogeneous Models**

Examples:

- Integrated execution of multiple disparate water/rainfall models
- Integration of infrastructure models, e.g., electric grid & economic networks

- **More Accessible Parallel Platforms**

- **Low-end**

Almost every end-user computing device having multiple cores

- **Medium-scale**

Desktops to have dozens/hundreds of cores

- **High-end**

Number of installations with 512 processors or more is increasing

- **Better Understanding of PADS**

- **Almost 3 decades of research**

- **Technology ready to be leveraged and applied**

- Time Advances & Timelines
- Orderings
- Spatial Parallelism
- Time Parallelism
- Execution Granularity
- Abstractions or Canonical Views

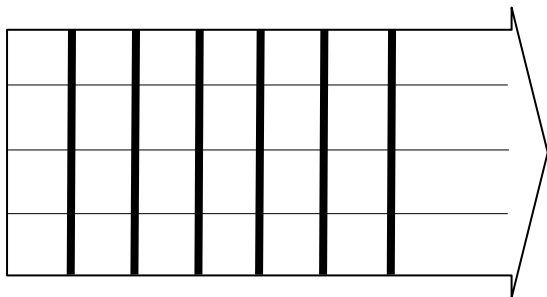
# Simulation Time Advancement Methodologies

- **Time-Stepped**

- All entities are paced with time increment  $\Delta t$
- Entities exchange state updates via messages

*Can be considered a canonical model: all others can be reduced to this*

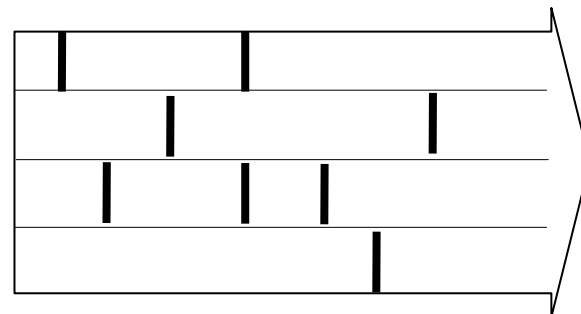
→ Simulation time



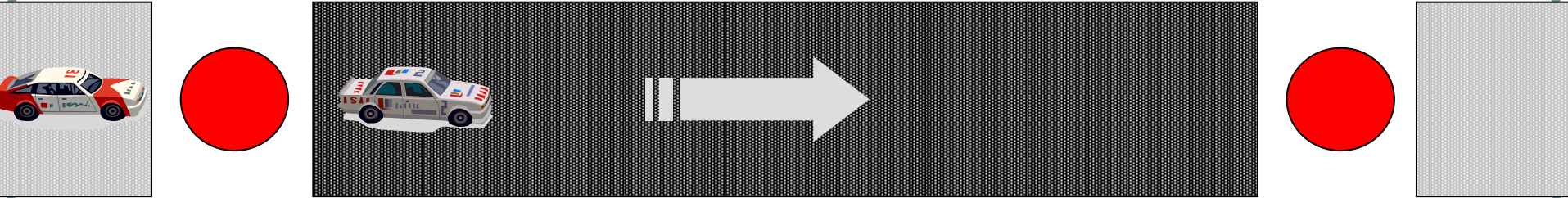
- **Discrete-Event**

- State updates are scheduled at different times in the future
- Entities exchange **events** for state updates
- Events are executed in timestamp order

→ Simulation time



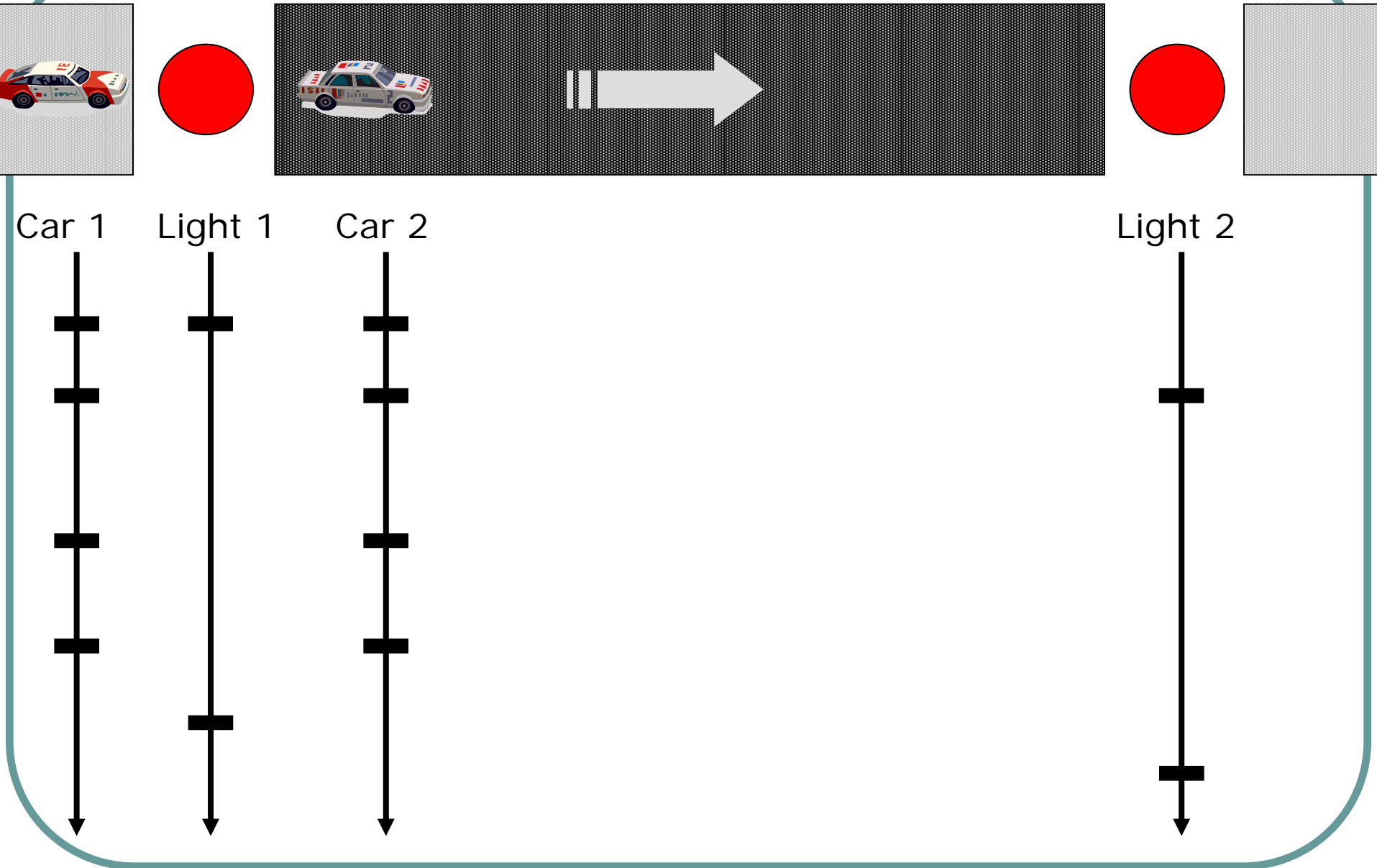
# A Simple Example



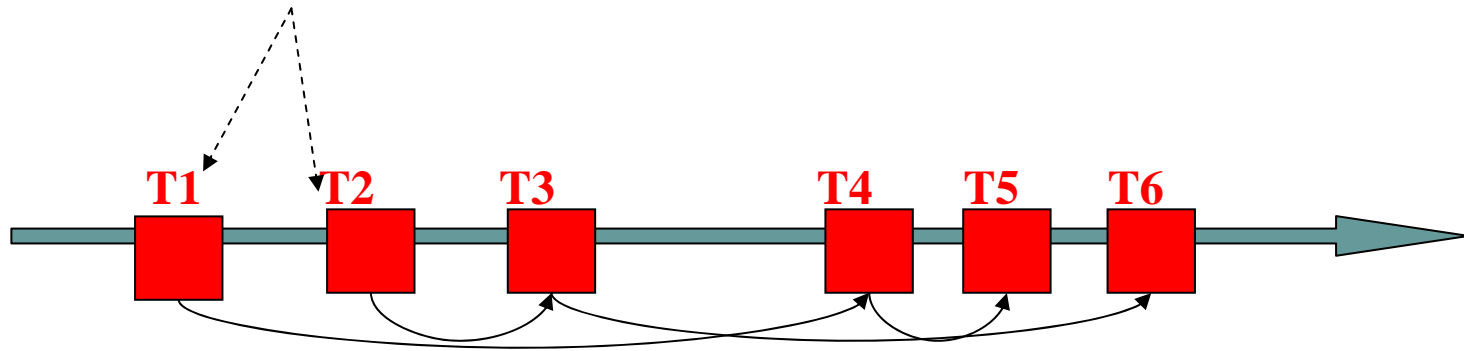
How is this executed in time-stepped models?  
How is this executed in discrete event models?



# Discrete Event Model – Example



Time Stamped Events



Simulation Time  
Progressing Left to Right

$T1 \geq T2$

$T2 \geq T3$

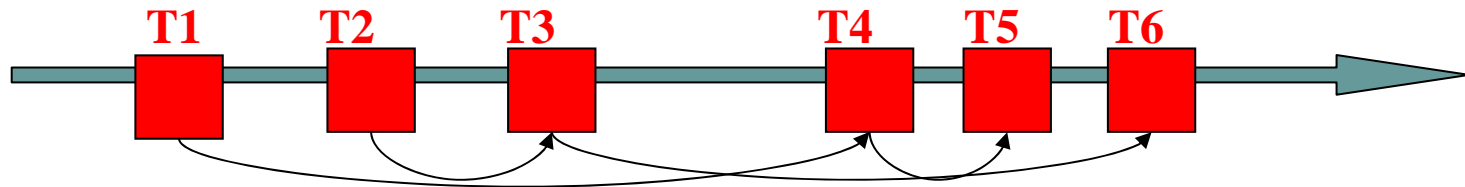
...

# Sequential & Parallel Discrete Event Simulation

## Sequential

1 processor

Example:

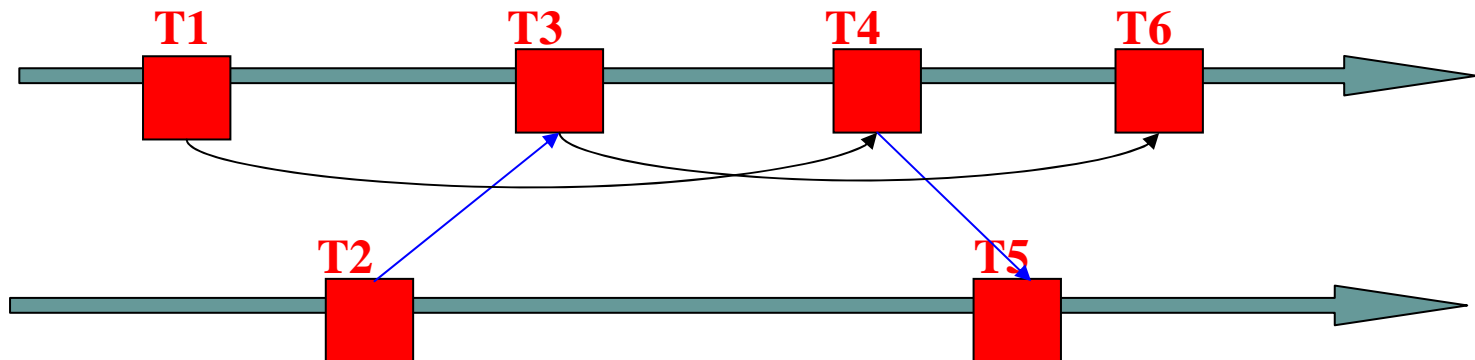


## Parallel

$n > 1$  processors

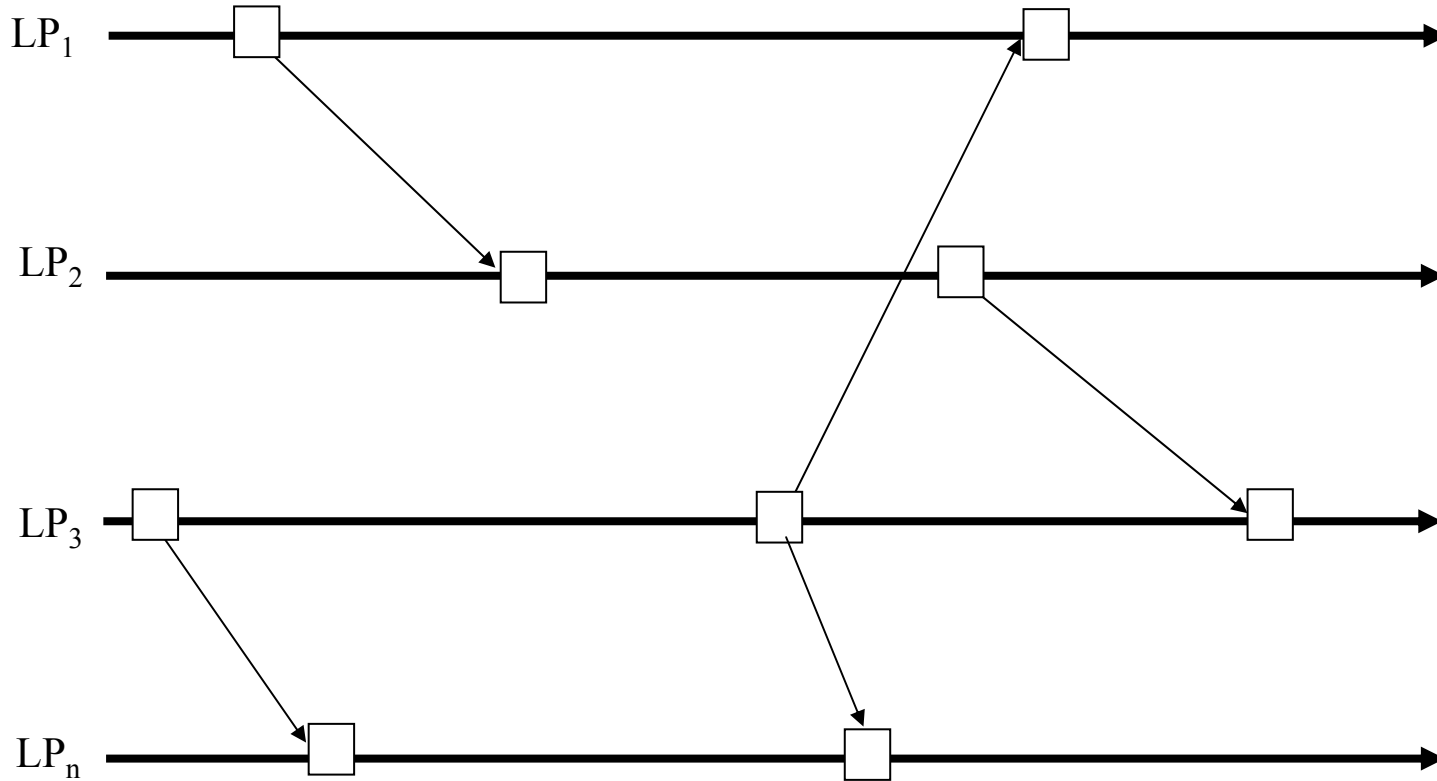
Example:

2 processors



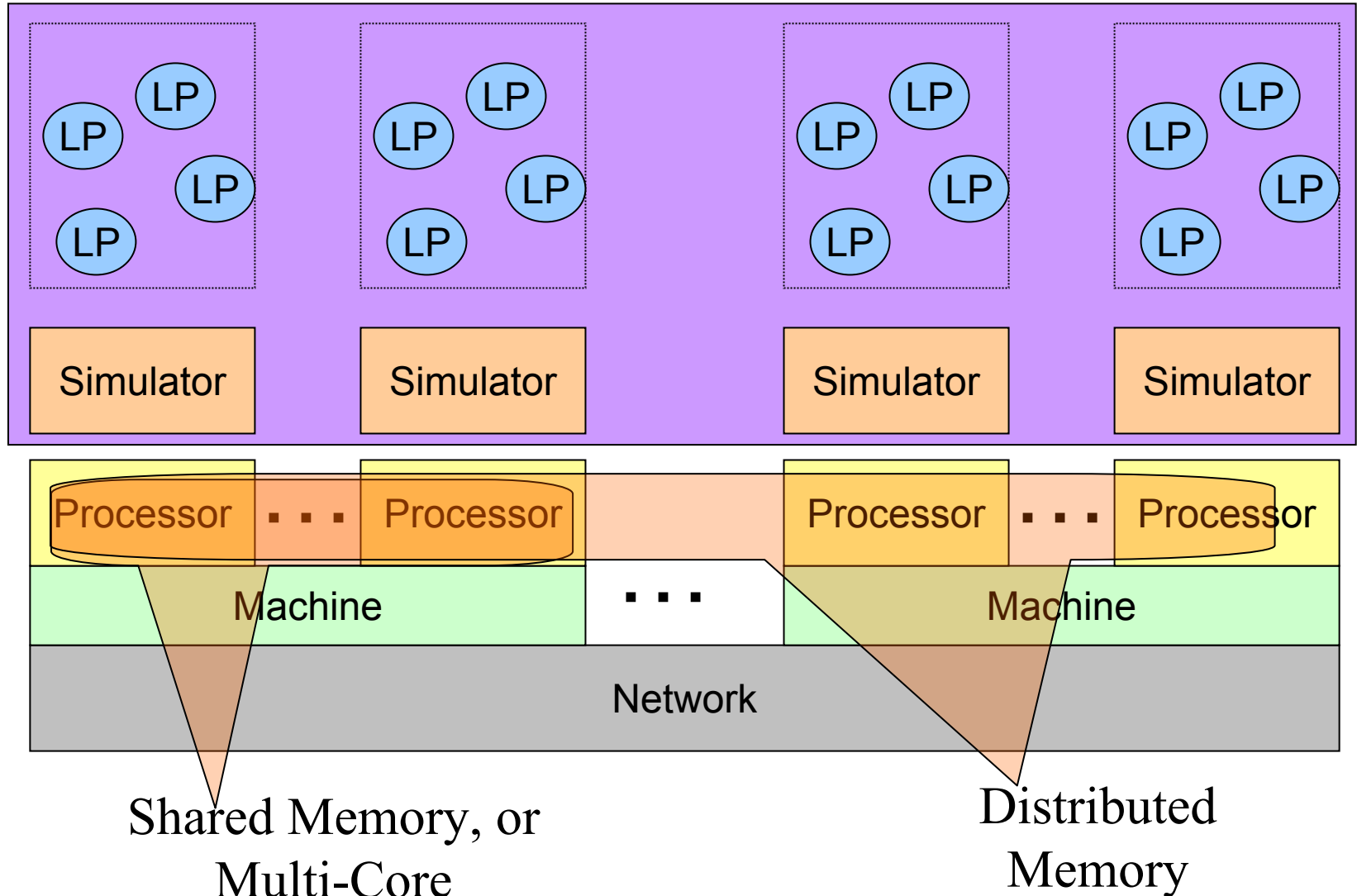
# Timelines: Parallel Discrete Event Simulation View

LP = Logical Process

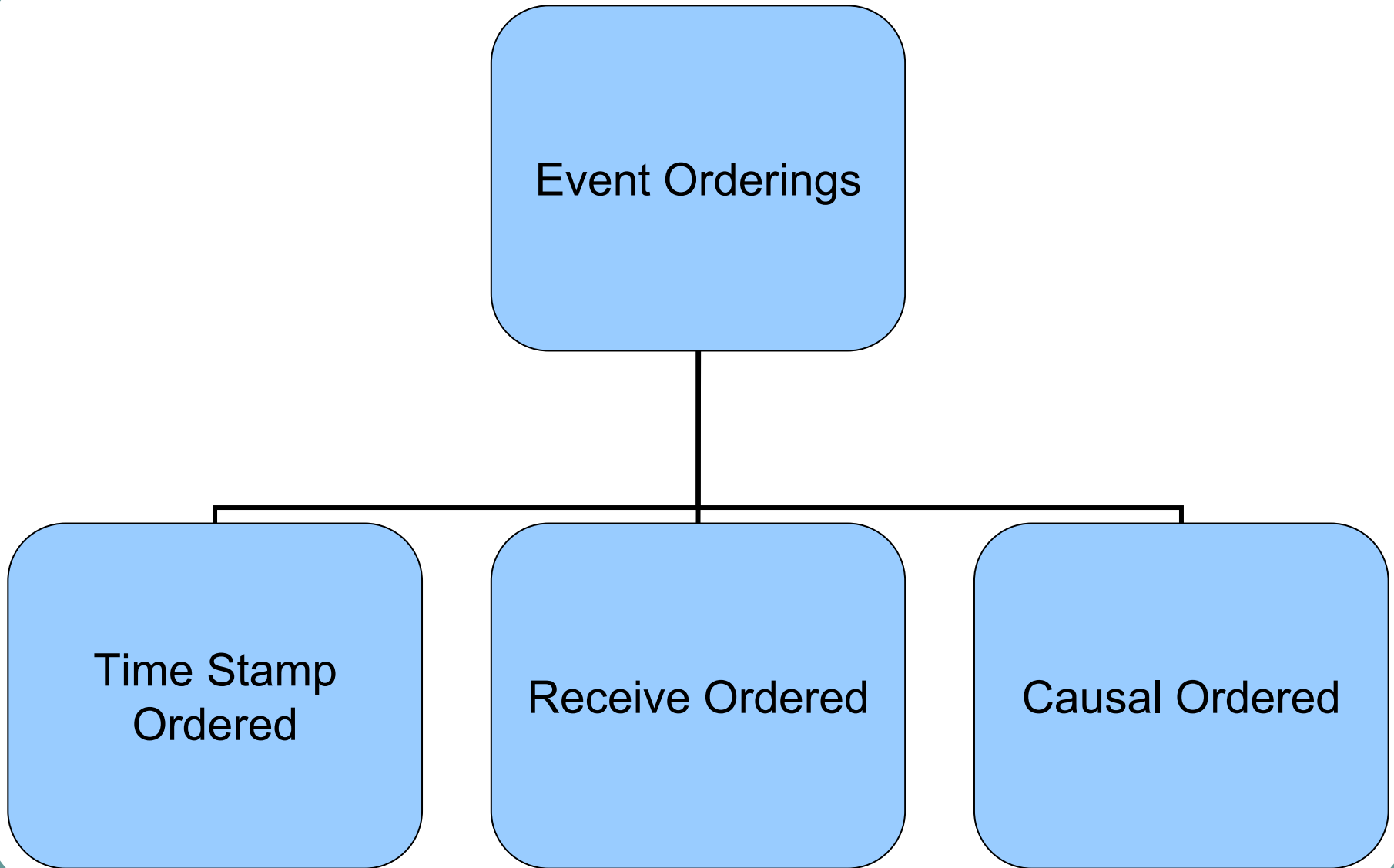


# Abstraction of Typical Parallel Execution Architecture

LP=Logical Process with its own timeline



# Event Orderings



- **Time Stamp Ordered**
  - Events are executed strictly in non-decreasing time stamp order
  - All processors see exactly the same order (globally ordered)
  - Repeated executions are deterministically reproducible
- **Receive Ordered**
  - Events are executed in the order of their arrival
  - If more than one event exists, they are executed in time stamp order
  - Repeated executions do not guarantee same results
- **Causal Ordered**
  - Causal chain of precedence is maintained
  - However, non-causally related events are executed in arbitrary order
  - Execution order of non-causally ordered events can change across processors

# Timestamp-Ordered Processing Services

PADS synchronization techniques address two fundamental issues in parallel/distributed simulation systems:

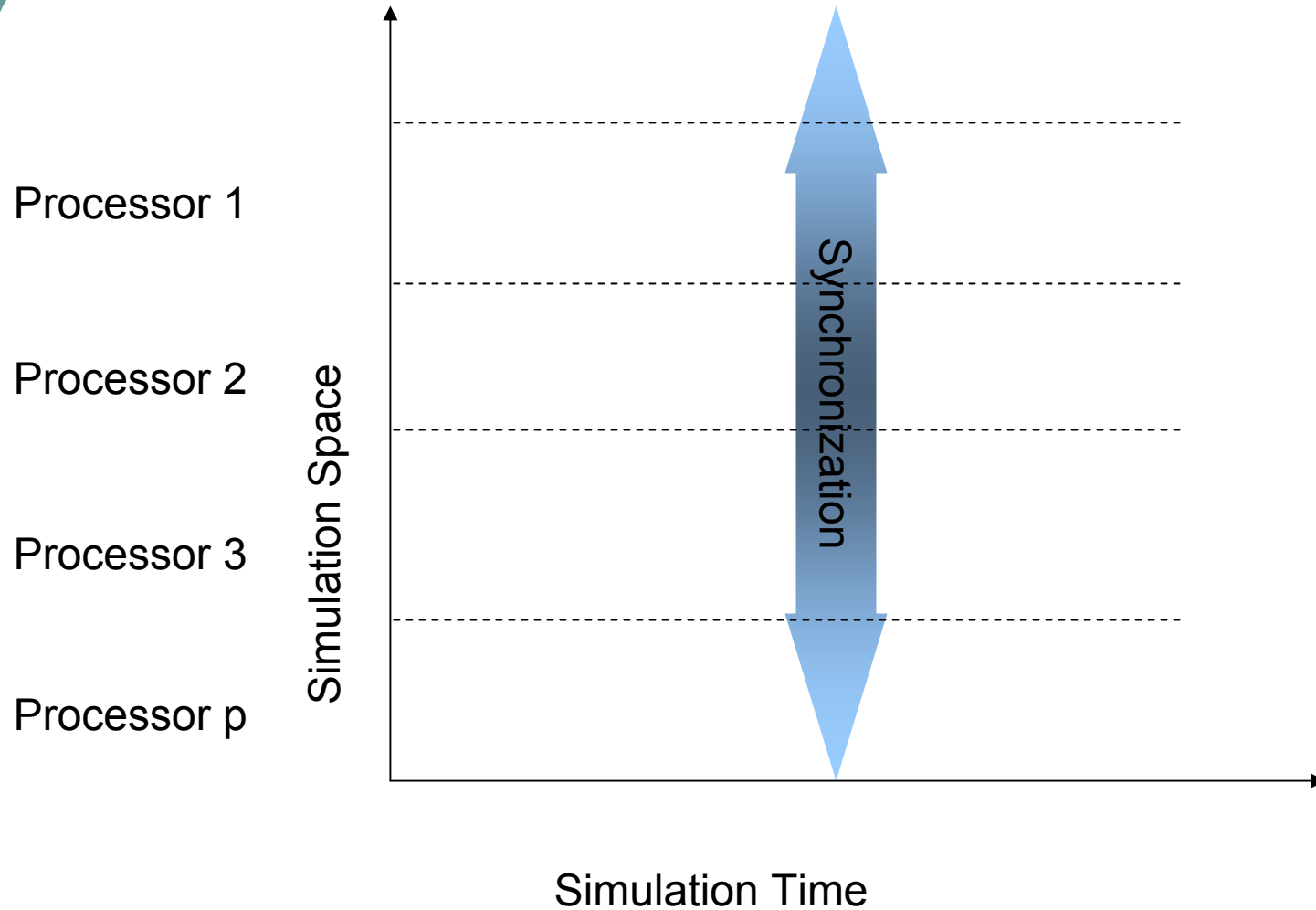
- **Event order**

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

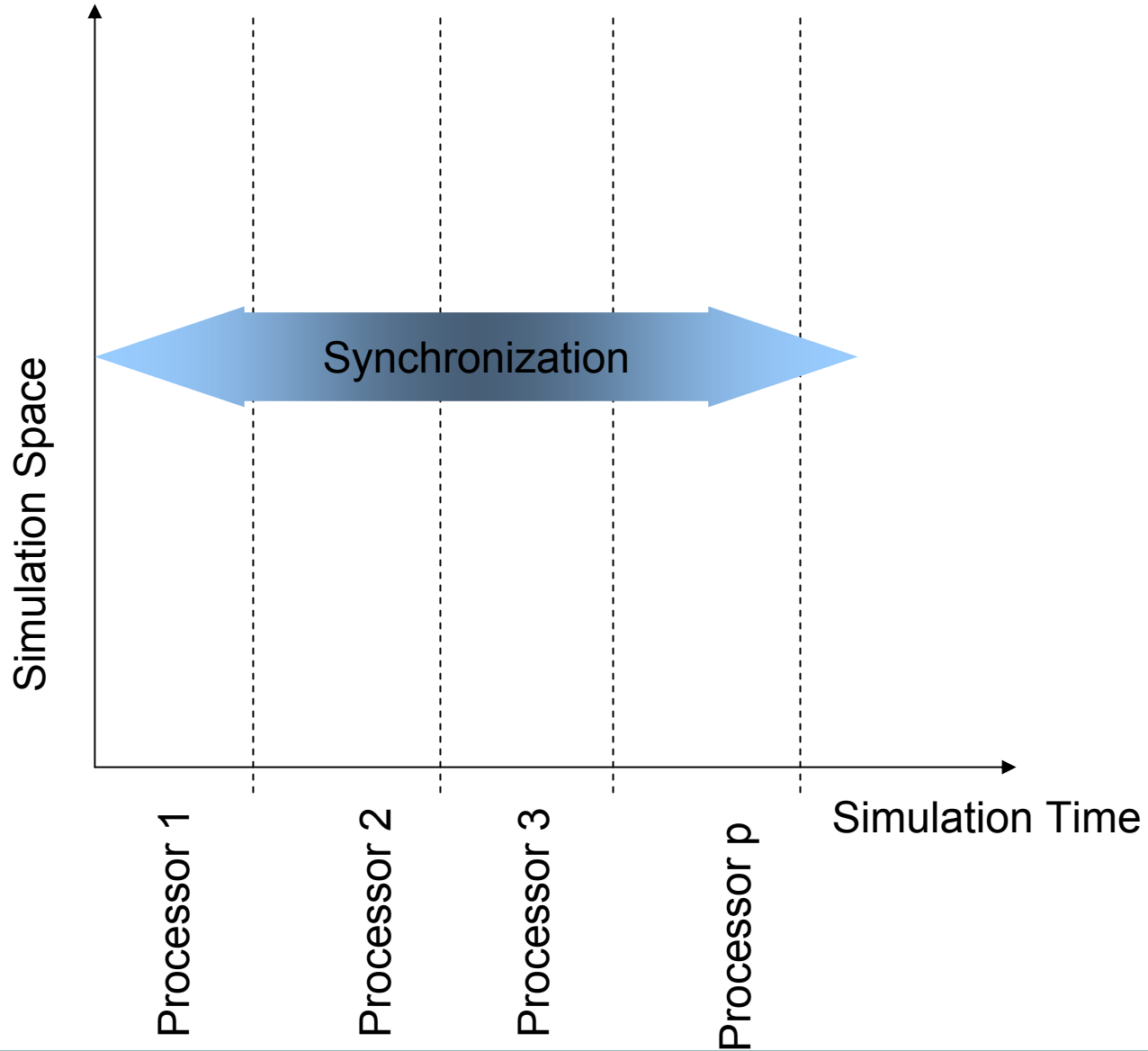
- **Time synchronized delivery**

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

# Space Parallel Method



# Time Parallel Method



# Execution Granularity

- **Very Fine-Grained**
  - 1 – 10  $\mu$ s
  - E.g., Digital Circuit Simulations
- **Fine-Grained**
  - 10 – 100  $\mu$ s
  - E.g., Packet-level TCP/IP Network Simulations
- **Medium-Grained**
  - 100 – 1000  $\mu$ s
  - E.g., Agent-Based Simulations
- **Coarse-Grained**
  - 1 – 10 ms
  - E.g., Battle-field Simulations
- **Importance**
  - Granularity directly determines PADS techniques' effectiveness

# Dimensions of PADS

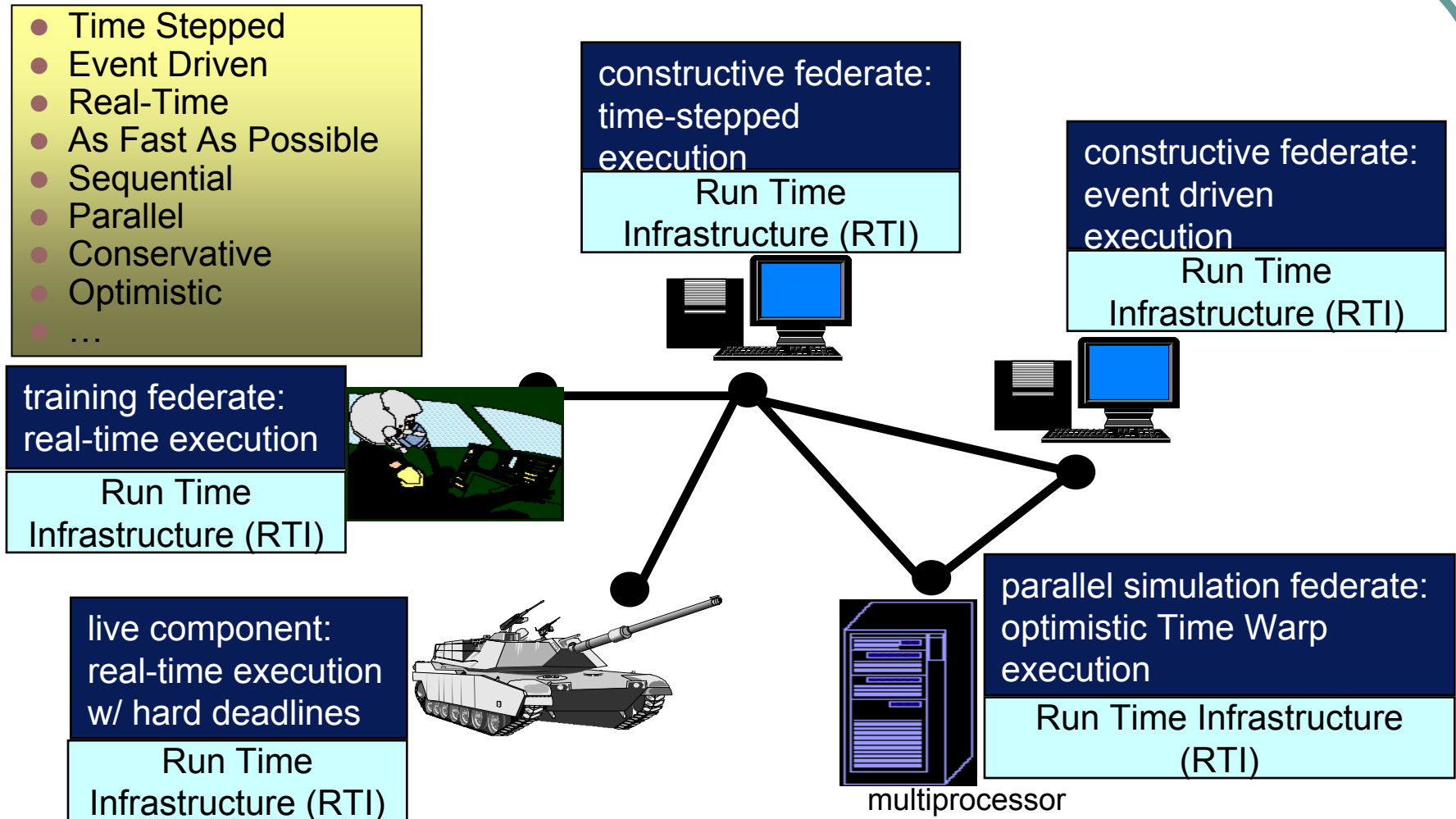
- Scheduling & Synchronization
- Interoperability
- Computation Platforms
- Composition Methods
- Interface & Implementation

PADS Research spans the Cross Product of above

- Syntactic
- Model Semantic
- Multi-scale Temporal
- Multi-type Pacing

# Integrating Multiple Types of Pacing

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



**Goal:** Provide services for interoperability among simulators with different local time pacing schemes in a single federation execution

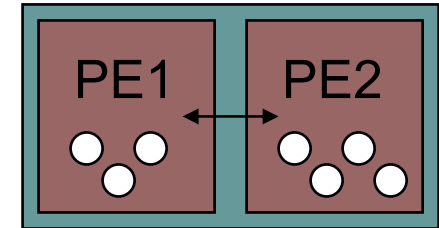
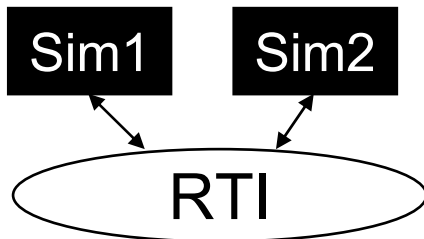
# Composition Methods

- Monolithic
- Federated
- Ad Hoc

# Federated vs. Monolithic Composition

Coarse-grained units

Fine-grained units



## Black-box Approach

- Minimal application modifications
- Autonomous simulators
- Coarse-grained concurrency control
- Widely used in defense interactive simulations
- E.g. HLA, DIS, ALSP

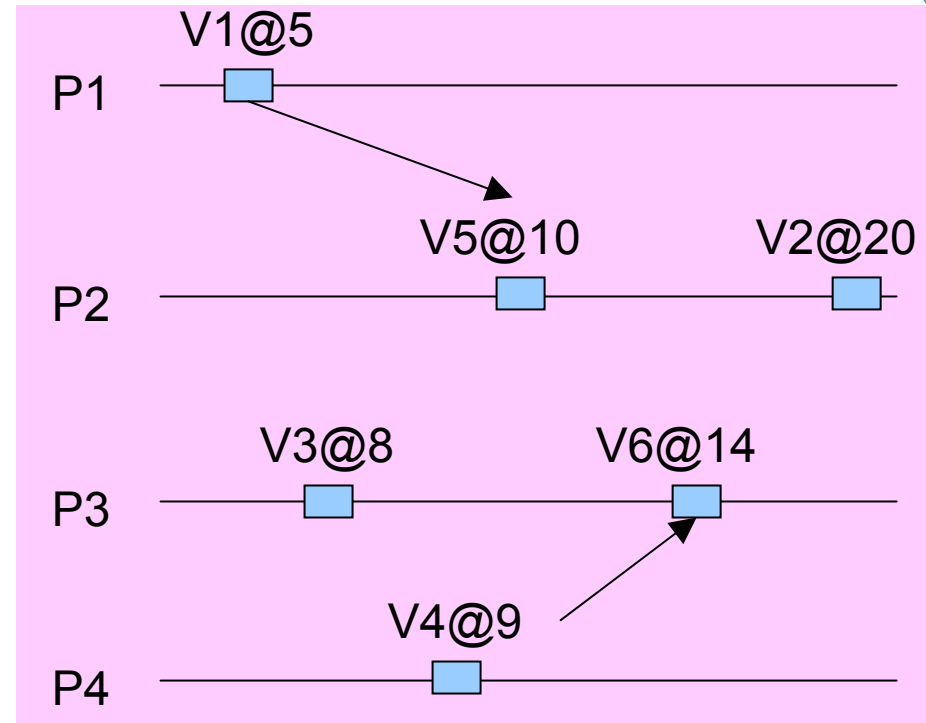
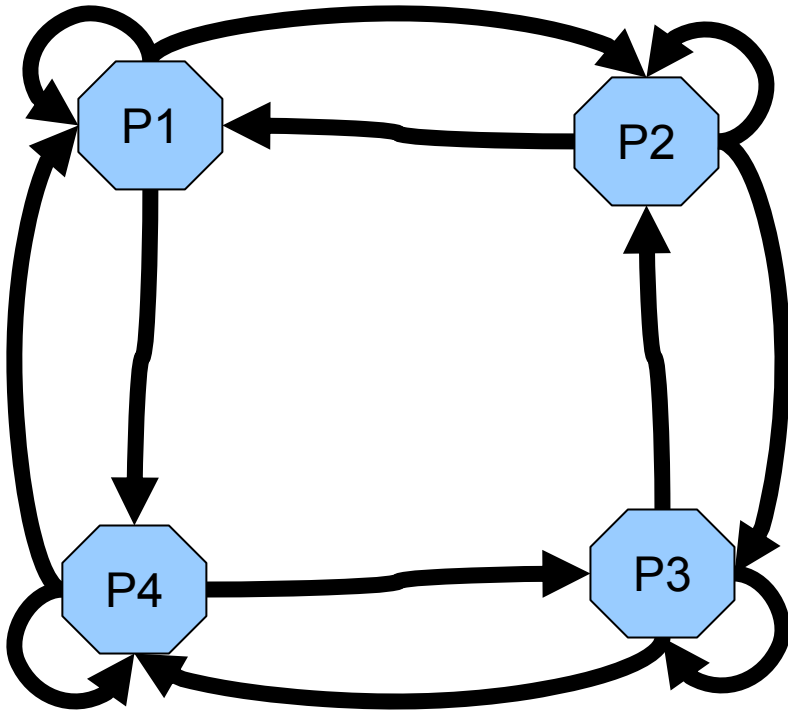
## Stand-alone Approach

- Integration can require significant modifications
- Homogeneous
- Fine-grained concurrency control
- Restricted languages, libraries, tools
- E.g. TeD, SPEEDES, PARSEC, SSF, Task-Kit

- Using naturally available mappings across models
  - Minimizes *a priori* model integration effort
- Example: Mobility + Communication in Vehicular Network Simulation
  - Map position updates from mobility simulator to wireless network simulator
  - Map communication effects from wireless network simulator to corresponding vehicles in mobility simulator
  - PADS'07

- Libraries  
E.g., libSynk
- Languages  
E.g., Maisie, TeD
- Frameworks  
E.g., HLA, XMSF, SSF
- Development Environments  
E.g., Commercial HLA Packages, MATLAB

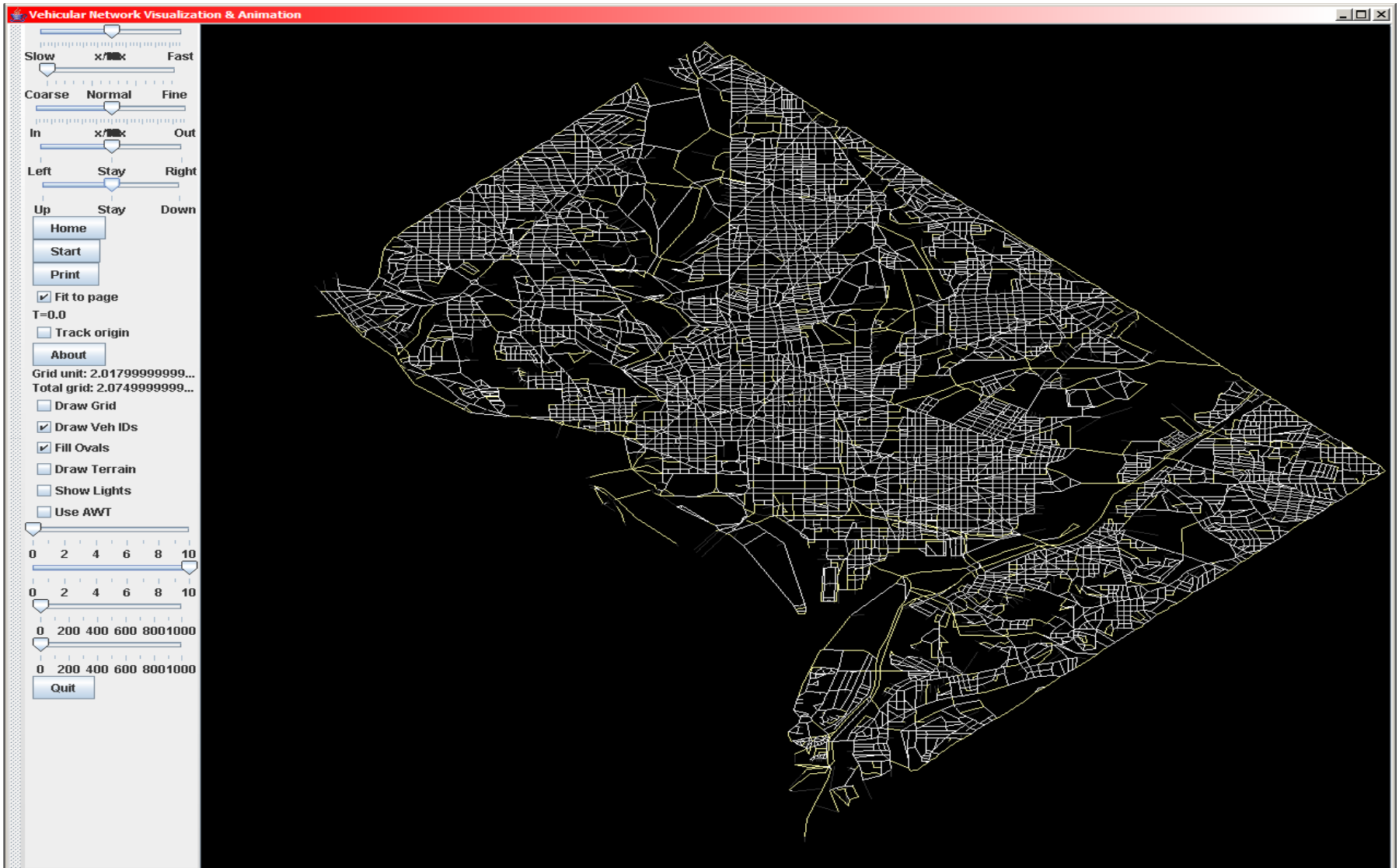
# Example - Simple Traffic Simulation



- Intersections P1..P4 have two-way links
- Vehicles arriving at intersection are parked for random period  $0..dt$
- Vehicles take 5 seconds to move between intersections

*Note: Every processor maintains a priority queue of "future events"*

# Test Network: Washington D.C.



# Targeted State Scale (e.g., Tennessee)

**Vehicular Network Visualization & Animation**

Slow x/100x Fast

Coarse Normal Fine

In x/100x Out

Left Stay Right

Up Stay Down

Home

Start

Print

Fit to page

T=0.0

Track origin

About

Grid unit: 0.0010km x 0.0010km  
Total grid: 0.01km x 0.01km

Draw Grid

Draw Veh IDs

Fill Ovals

Draw Terrain

Show Lights

Use AWT

0 2 4 6 8 10

0 2 4 6 8 10

0 200 400 600 800 1000

0 200 400 600 800 1000

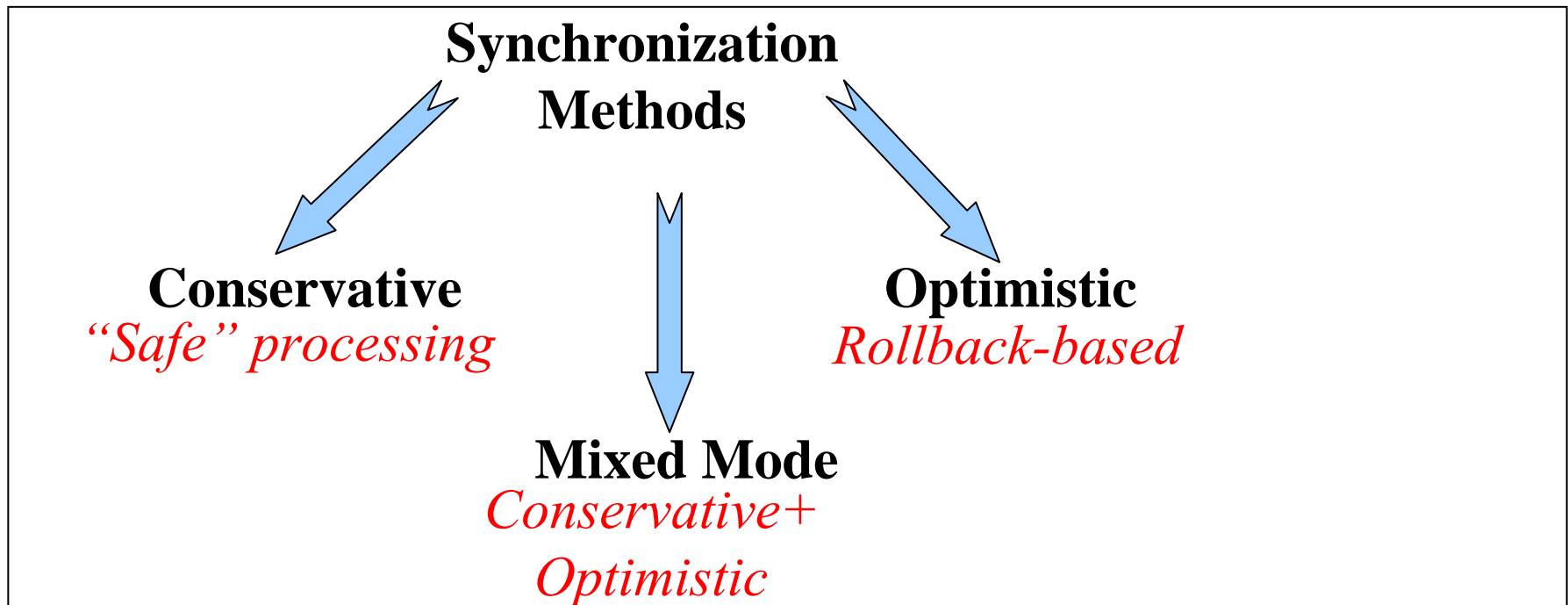
Quit

# Traditional Techniques

- Centralized Coordination
- Conservative Parallel
- Optimistic Parallel
- Real Time Parallel

# Broad Classification

**Goal:** Ensure global timestamp-ordered processing.  
=> Synchronization among simulators required.



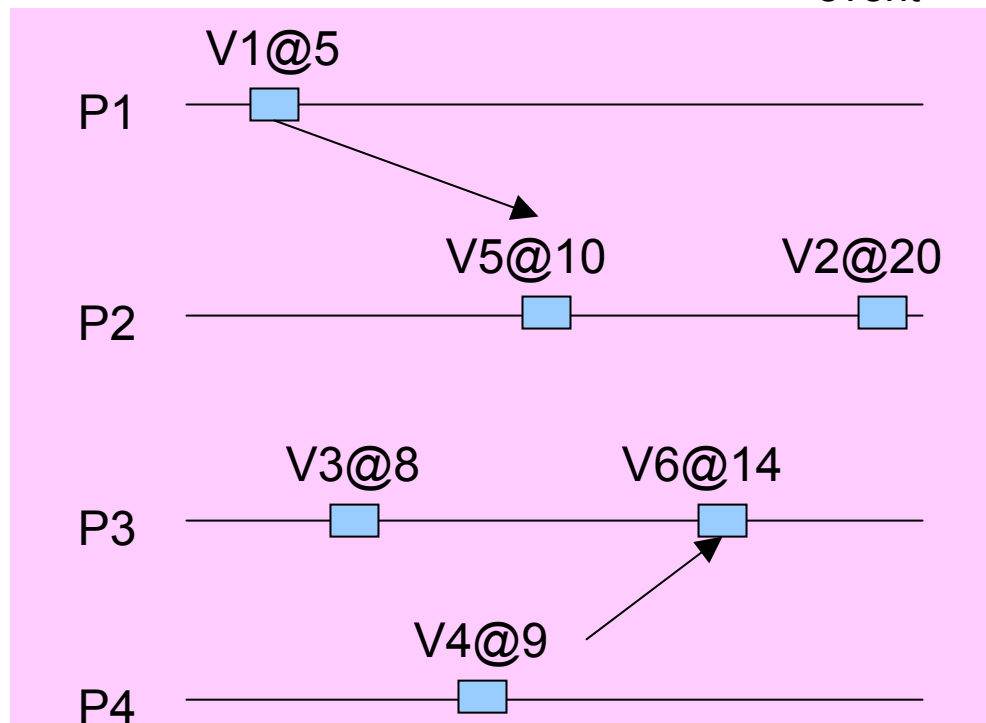
# Synchronization Approaches

- Conservative
  - Avoid synchronization errors (message in processor's past)
  - Use blocking to avoid errors
- Conservative processing offers ease of implementation, but relies on “lookahead”
- 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

# Centralized Coordination

Algorithm: Execute the following repeatedly

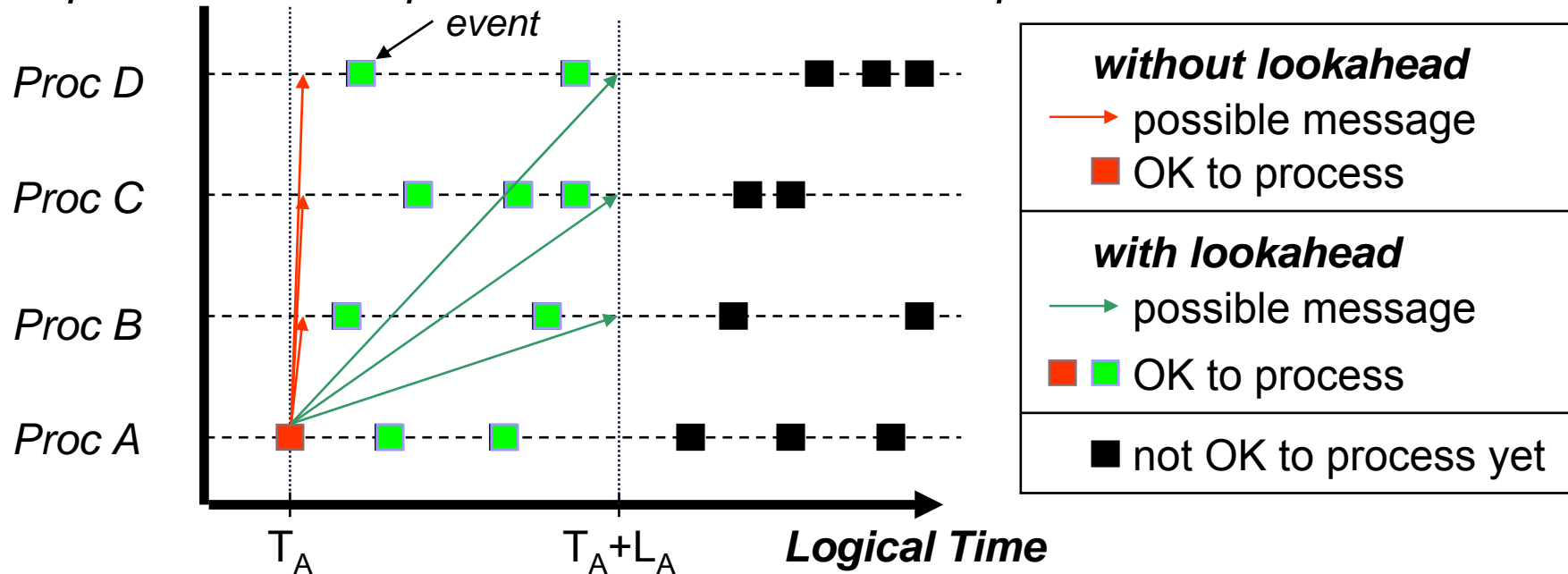
- $(P_{\min}, T_{\min})$  = Find processor with minimum timestamp
- $(P_{\min 2}, T_{\min 2})$  = Find processor with second minimum
- Advance  $P_{\min}$  at most up to time  $T_{\min 2}$  or up to  $T_{\text{event}}$  if  $P_{\min}$  sends an event with timestamp  $T_{\text{event}} < T_{\min 2}$



# Lookahead Solution for Concurrency Problem\*

*Problem: Limited concurrency in event driven simulators*

*Each processor must process events in time stamp order*



Processor A declares a lookahead value  $L_A$ ; the time stamp of any event generated by the Processor 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)

# Conservative Parallel Execution Template

- $T_{\min}$  = Min timestamp in future event list
- Do the following repeatedly
  - Evaluate lower bound of incoming timestamp LBTS, with my own guarantee of  $(T_{\min} + \text{lookahead})$  to others
  - While(  $T_{\min} \leq \text{LBTS}$  )
    - Dequeue the event with timestamp  $T_{\min}$
    - Execute that event
    - $T_{\min}$  = Min timestamp in future event list

How do we compute LBTS (lower bound on incoming timestamp)?

- Null Messages
- Global Reductions

- Time Warp
- Data Structures
- Rollback
- State Saving
- Shared Memory Optimizations

# An Optimistic Algorithm – Time Warp\*

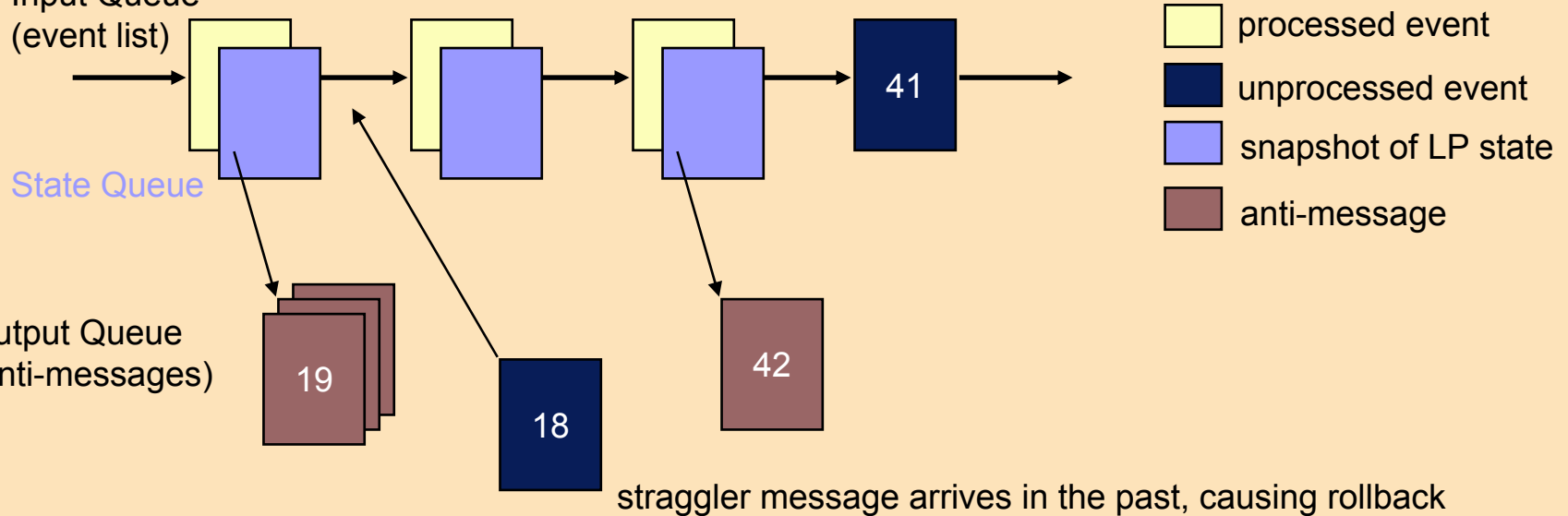
Each processor executes its events in time stamp order, like a sequential simulator, except it: (1) does NOT discard processed events and (2) adds a

rollback mechanism

Input Queue  
(event list)

State Queue

Output Queue  
(anti-messages)

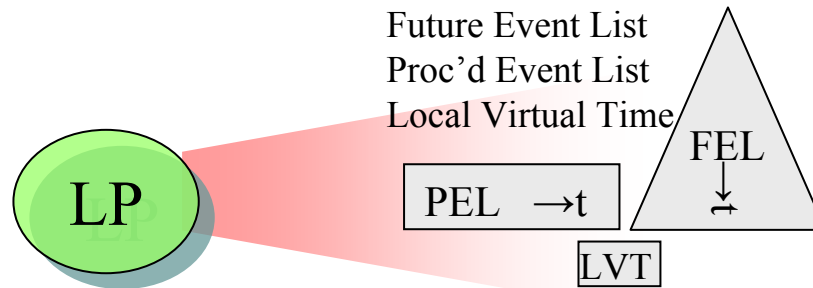


Adding rollback:

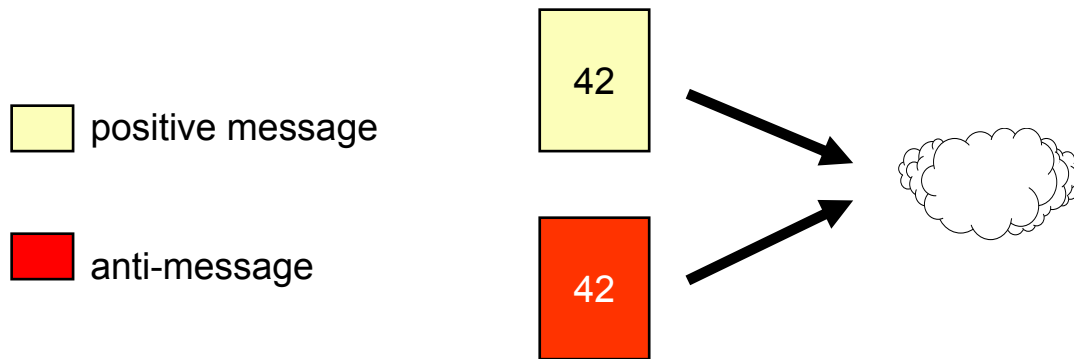
- a message arriving in the processor's past initiates rollback
- to roll back an event computation we must undo:
  - changes to state variables performed by the event;  
*solution: checkpoint state or use incremental state saving (state queue)*
  - message sends  
*solution: anti-messages and message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

Every LP contains the following:

- Future Event List (Priority Queue)
- Processed Event List (deque)
- Local Virtual Time



# Anti-Messages In Optimistic Federates



- Used to cancel a previously sent message
- Each positive message sent by a processor 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 processor 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.

# Synchronization Algorithm Implementation

- **Conservative Synchronization**
  - Local communication-based, e.g., Chandy-Misra-Bryant, YAWNS
  - Global communication-based, e.g., Consistent cut, Flush Barrier, Reduction
  - All provide a “lower bound on incoming events’ time stamps” (LBTS)
- **Optimistic Synchronization**
  - Shared memory-based, e.g., Hybinette-Fujimoto
  - Distributed memory-based, e.g., Iterated Global Reductions, Mattern’s
  - All compute “global virtual time” (GVT) or equivalent
- **Software Implementations**
  - Implementation is relatively complex
  - Reusable libraries are available, e.g., libSynk,  $\mu$ sik
  - Standard run-time infrastructures (RTI) available, e.g., HLA RTI

- Local Rollback
- Remote Rollback

- Copy State Saving
  - Save entire LP state before every event
- Periodic State Saving
  - Save entire LP state before every  $p$  events
- Incremental State Saving
  - Save only modified parts of LP state for every event

# Shared Memory Optimizations

- **Fast Global Virtual Time**
  - Since memory is globally accessible by all processors, transient events are easily detected by synchronization with global variables
- **On-the-fly Fossil Collection**
  - Events are enqueued on “potentially free” list
  - Those events whose receive timestamp is less than GVT are reused
  - Thus, no explicit fossil collection step required

- Distributed Interactive Simulation
- High Level Architecture
- Real Time Clocks
- Network Time Protocol

- Alternative Platforms
- Supercomputing-based
- Asynchronous Computing
- Alternative Modeling
- Standards
- Massively Multiplayer Gaming
- Alternative Synchronization
- Branching Multiple Simulations via Cloning

# Alternative Platforms

- Network Coprocessors
- Graphical Processing Units (GPGPUs)
- Hybrid Processors (Cell)
- Multi-Core

# Network Coprocessors

- Data Distribution Filtering
- State Saving
- Global Virtual Time

# Graphical Processing Units (GPGPUs)

- Time Stepped Simulation
- Line of Sight Calculations
- Discrete Event Simulation

# Generalized Programmability for Complex Graphics

## Vertex Shader Source Code for Bump-Reflection Mapping



Figure 15 Example of Bump-Reflection Mapping

```
struct a2v {
    float4 Position : POSITION; // in object space
    float2 TexCoord : TEXCOORD0;
    float3 T : TEXCOORD1; // in object space
    float3 B : TEXCOORD2; // in object space
    float3 N : TEXCOORD3; // in object space
};

struct v2f {
    float4 Position : POSITION; // in projection space
    float4 TexCoord : TEXCOORD0;

    // first row of the 3x3 transform
    // from tangent to cube space
    float4 TangentToCubeSpace0 : TEXCOORD1;

    // second row of the 3x3 transform
    // from tangent to cube space
    float4 TangentToCubeSpace1 : TEXCOORD2;
};
```

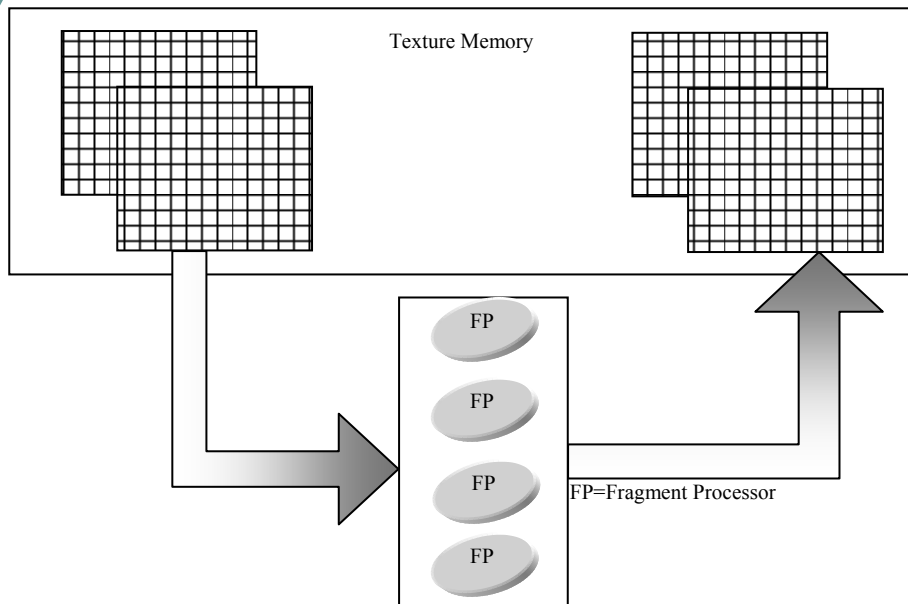
the various techniques can be combined to produce compelling, stylized skin.



Figure 10 Example of Skin

Examples from NVIDIA's Cg Toolkit

# GPGPU as Parallel Computer: Abstraction



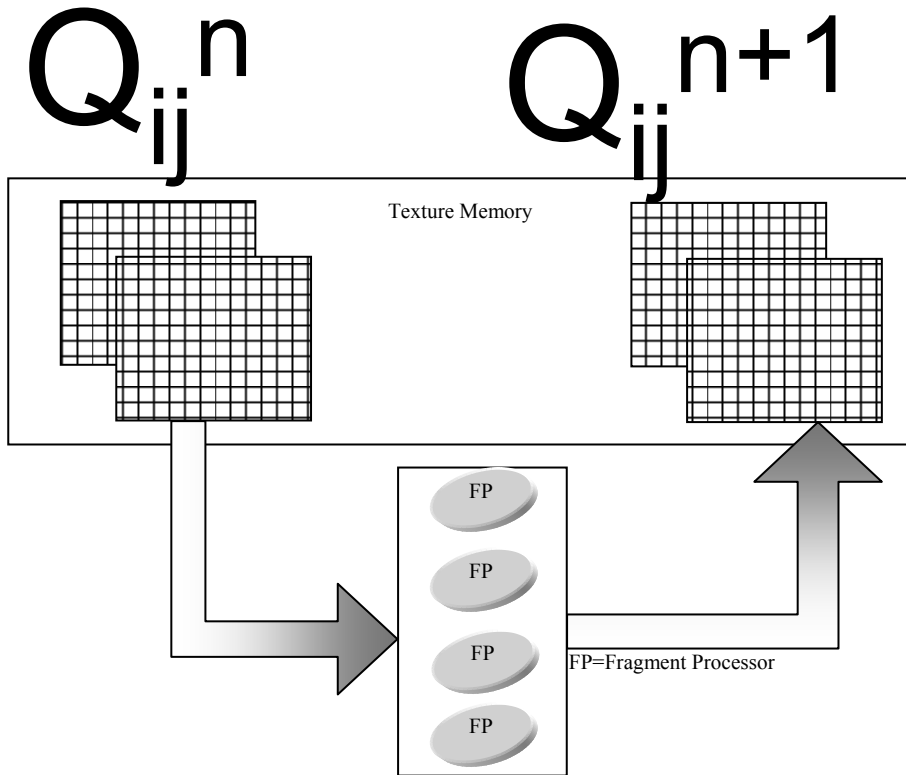
Textures = Memory variable arrays

Fragment processors = CPUs

Rendering = Computation

- GPGPU = General Purpose Graphical Processing Unit
- Multiple parallel processing units
- Closely connected via “shared memory”
- Very fast number crunching
- Built-in asynchronous memory operations, caching, etc.

# Mapping Time Stepped Update to GPGPU



$$\frac{\partial Q}{\partial t} = \alpha_x \frac{\partial^2 Q}{\partial x^2} + \alpha_y \frac{\partial^2 Q}{\partial y^2} + \beta$$

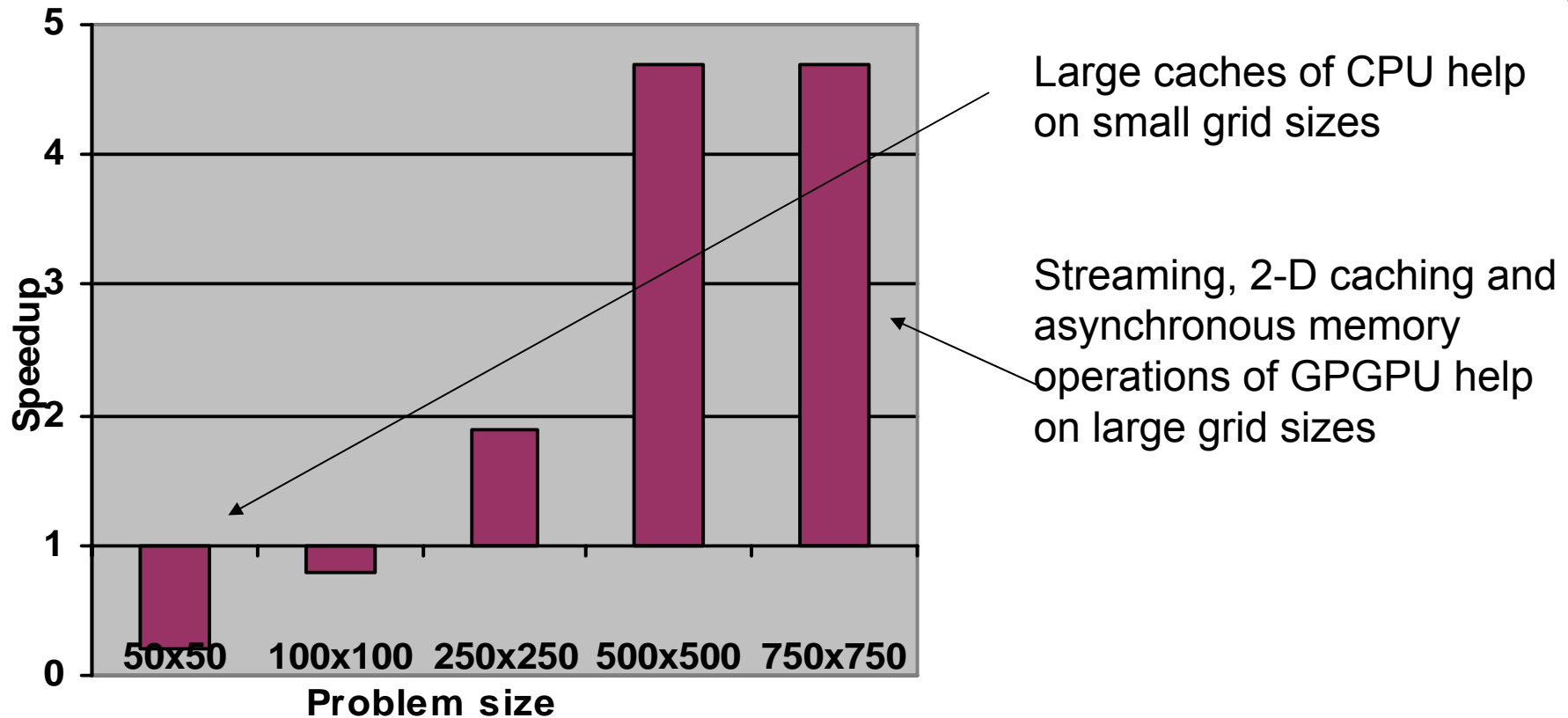
$$\frac{q_{i,j}^{n+1} - q_{i,j}^n}{\Delta t} = \alpha_x \frac{q_{i,j-1}^n - 2q_{i,j}^n + q_{i,j+1}^n}{\Delta x^2} + \alpha_y \frac{q_{i-1,j}^n - 2q_{i,j}^n + q_{i+1,j}^n}{\Delta y^2} + \beta$$

$$Q[i][j] = f( Q[i][j], \\ Q[i-1][j], Q[i+1][j], \\ Q[i][j+1], Q[i][j-1] )$$

Note: No read/write hazards

- Most existing GPGPU simulations are time-stepped!
- Shown to be much faster on GPGPU than on CPU

# Time-stepped Performance on GPGPU

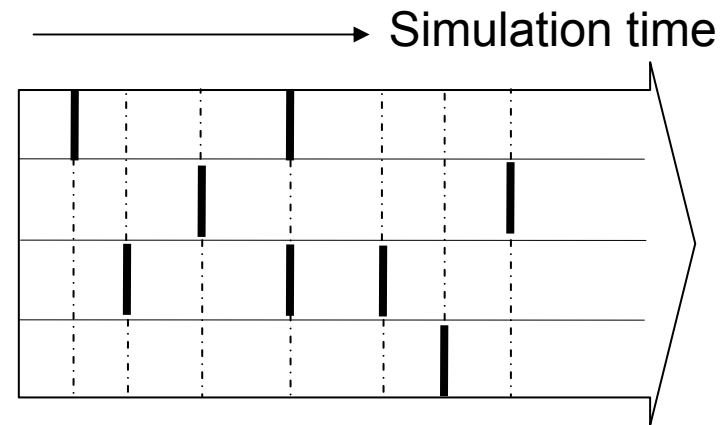


- Performance relative to time-stepped code on CPU
- 2x implies TS on GPGPU is twice as fast as on CPU

# Hybrid (Discrete + Time Stepped) Approach

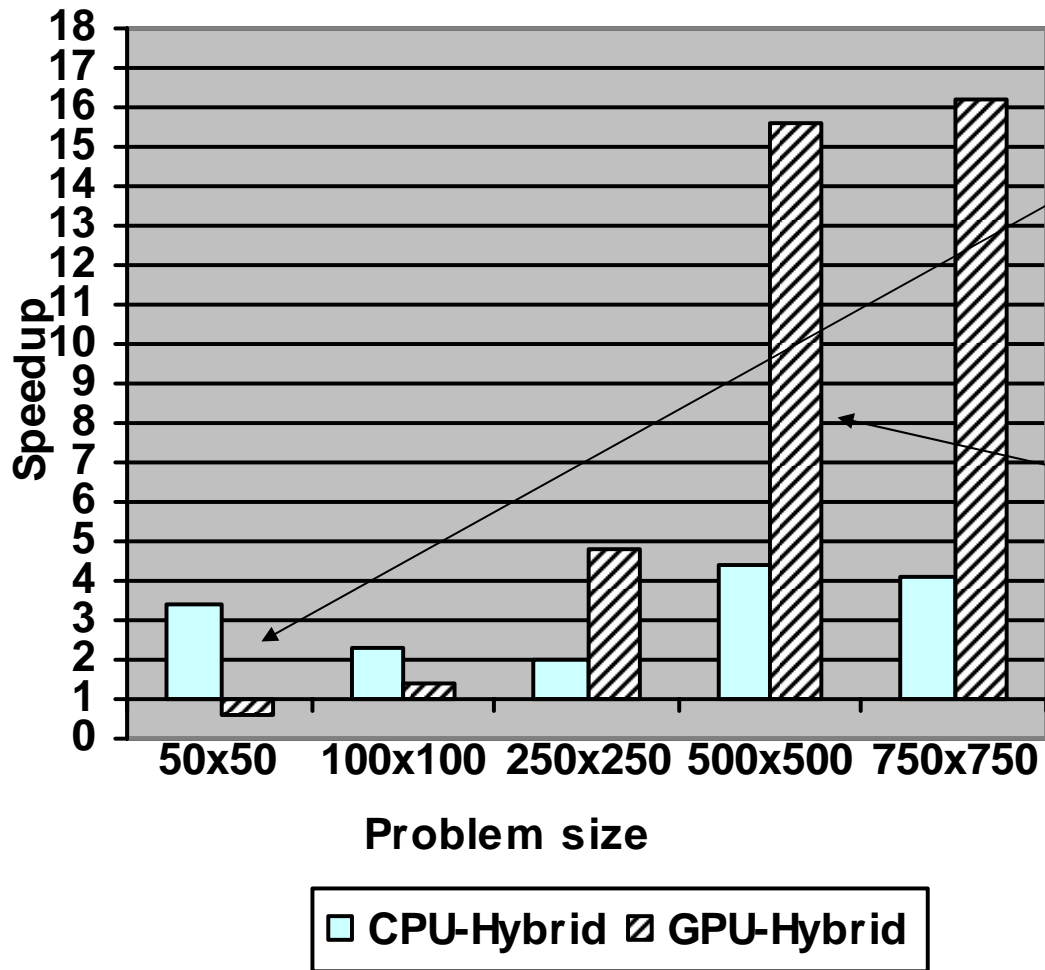
- Compute upper bound on  $\Delta t$  for each element
  - Solve for  $\Delta t$ , for a given resolution of  $Q$  (state space)
- Advance time by minimum  $\Delta t$ , update *all* elements
- Maps to GPGPUs very well!

$$\frac{\partial Q}{\partial t} = \alpha_x \frac{\partial^2 Q}{\partial x^2} + \alpha_y \frac{\partial^2 Q}{\partial y^2} + \beta$$



$$\frac{q_{i,j}^{n+1} - q_{i,j}^n}{\Delta t} = \alpha_x \frac{q_{i,j-1}^n - 2q_{i,j}^n + q_{i,j+1}^n}{\Delta x^2} + \alpha_y \frac{q_{i-1,j}^n - 2q_{i,j}^n + q_{i+1,j}^n}{\Delta y^2} + \beta$$

# Hybrid (Discrete + Time Stepped) Performance on GPGPU vs. CPU



CPU much faster due to very large cache

- Small grid fits in cache!

High gain on larger grids

- Faster time advances enabled by hybrid execution

- Performance relative to time-stepped code on CPU

- Performance gain from DES-style execution can be reaped on GPGPU as well
  - Using proper adaptation of DES to hybrid
- GPGPU can give several fold improvement over CPU performance on plain TS as well as DES (hybrid)
  - GPU-Hybrid is 17x relative to TS-CPU!

# Hybrid Processors (Cell)

- Shared Cache
- Shared Network I/O

# Supercomputing-based

- Progress
- Reductions-based
- Null Message-based
- Scalability Experiments

# Scaling Challenges

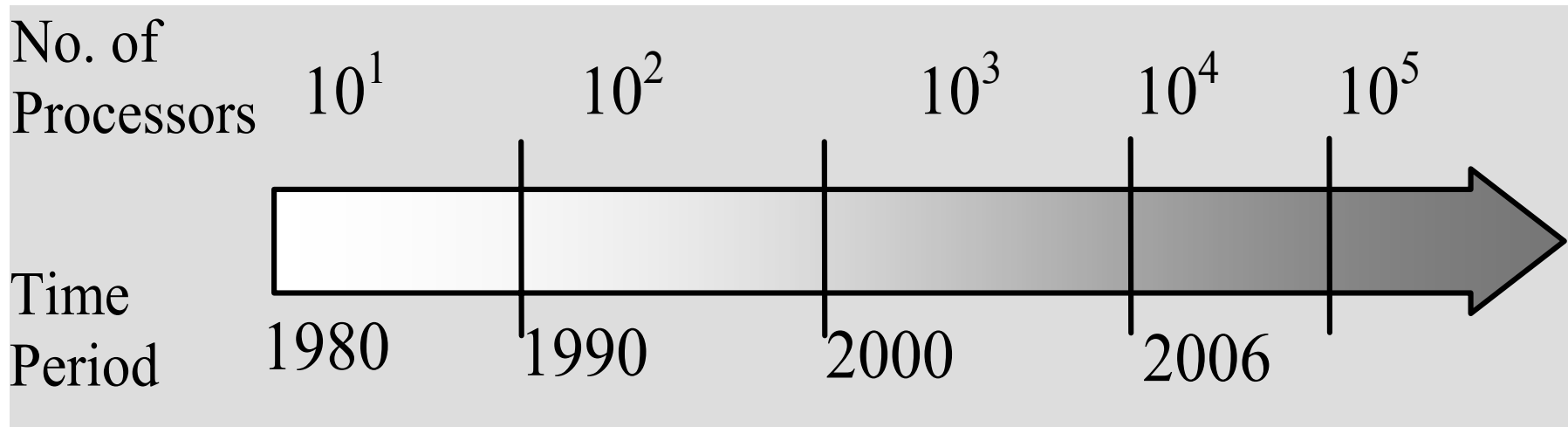
- Efficient global synchronization
  - Fast GVT/LBTS Computation
  - Smart/lazy sends, fast remote rollback
- Minimization of working set
  - Fast fossil collection
  - Fast anti-message reclamation
- Minimization of runtime overhead
  - Fast scheduling
  - Fast local rollback
- ...

# Synchronization Algorithm Implementation

- **Conservative Synchronization**
  - Local communication-based, e.g., Chandy-Misra-Bryant, YAWNS
  - Global communication-based, e.g., Consistent cut, Flush Barrier, Reduction
  - All provide a “lower bound on incoming events’ time stamps” (LBTS)
- **Optimistic Synchronization**
  - Shared memory-based, e.g., Hybinette-Fujimoto
  - Distributed memory-based, e.g., Iterated Global Reductions, Mattern’s
  - All compute “global virtual time” (GVT) or equivalent
- **Software Implementations**
  - Implementation is relatively complex
  - Reusable libraries are available, e.g., libSynk,  $\mu$ sik
  - Standard run-time infrastructures (RTI) available, e.g., HLA RTI

# PADS Progress: Supercomputing Scalability

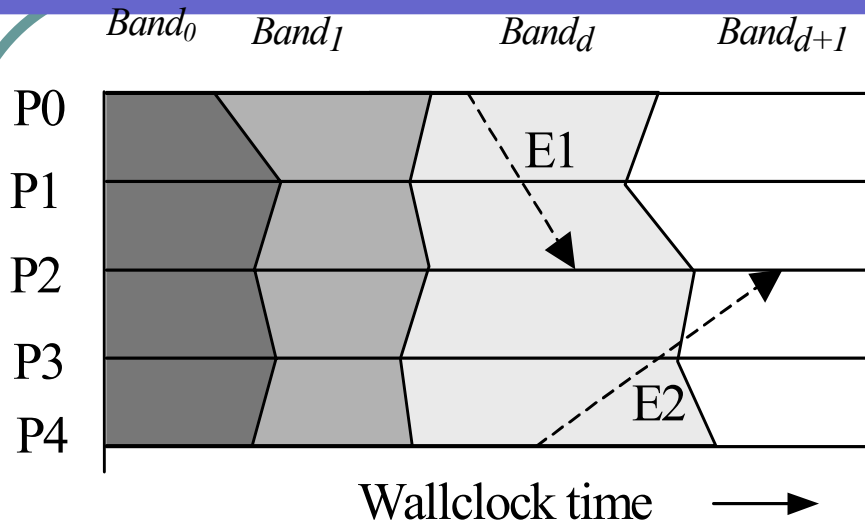
- Coming of age lately
- Time Warp & Mixed Mode possible now on  $10^3$ - $10^4$  processors  
Very recent (Oct 2006)



# Reductions-based Synchronization Algorithm - Features

- **Asynchronous**
  - Barriers not used
  - Simulation never paused for time management
  - Time management concurrent with simulation
- **On-demand**
  - Can be started by any processor at any time
  - Not necessary to have LBTS computation running always

# Terminology



At any processor in band  $d$

- Outgoing events marked with band  $d+1$
- Next band  $d+1$  entered only after receiving all events of band  $d$
- LBTS computation for band  $d-1$  is started and completed in band  $d$

For each band  $d$ , each processor  $i$  maintains  $\langle \tau_i, \delta_i \rangle$ :

$\tau_i =$  min unprocessed event timestamp

$\delta_i =$  total events sent – total events received  
(of events sent in band  $d$ )

Asynchronous distributed reduction (**trial**) used to compute:

$$\tau = \min(\tau_i)$$

$$\Delta = \text{sum}(\delta_i)$$

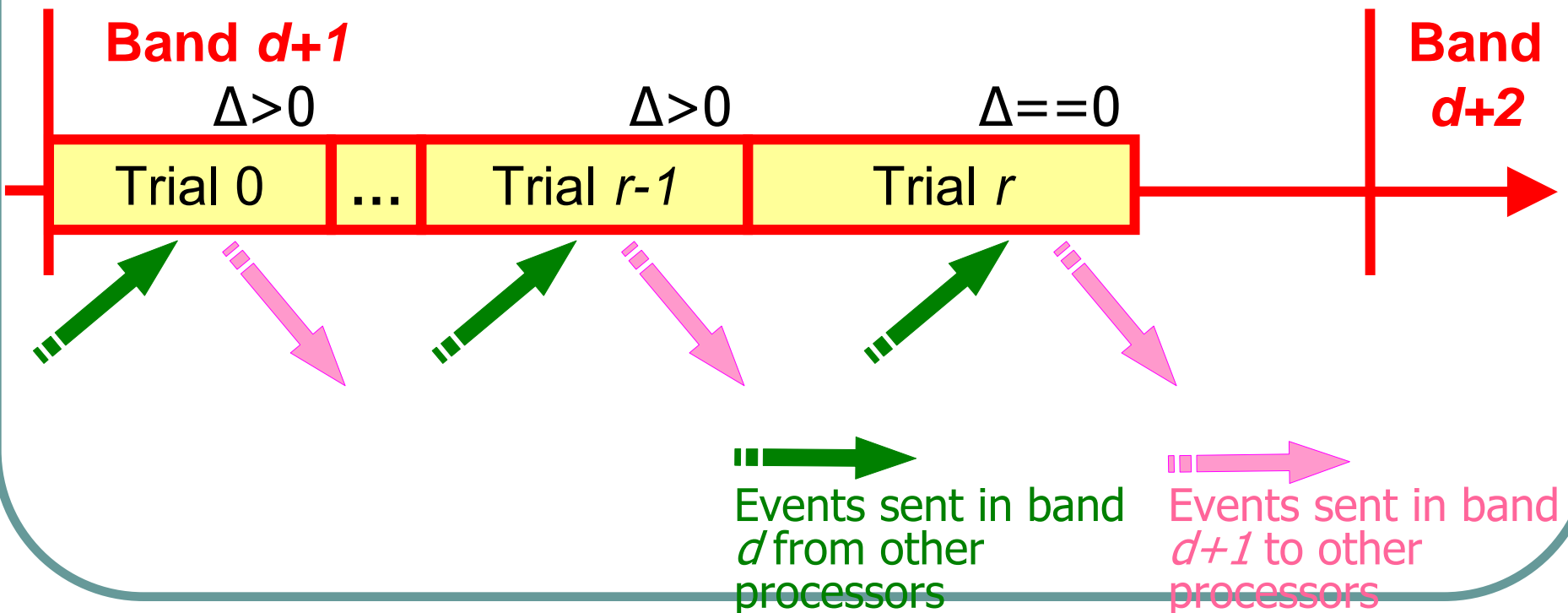
Note:  $\Delta == 0$  implies no more transient messages for band  $d$

$\Delta > 0$  implies some messages sent but not received yet

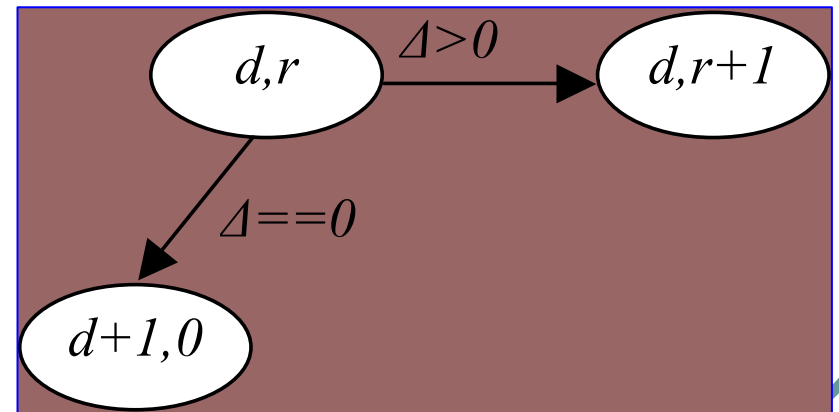
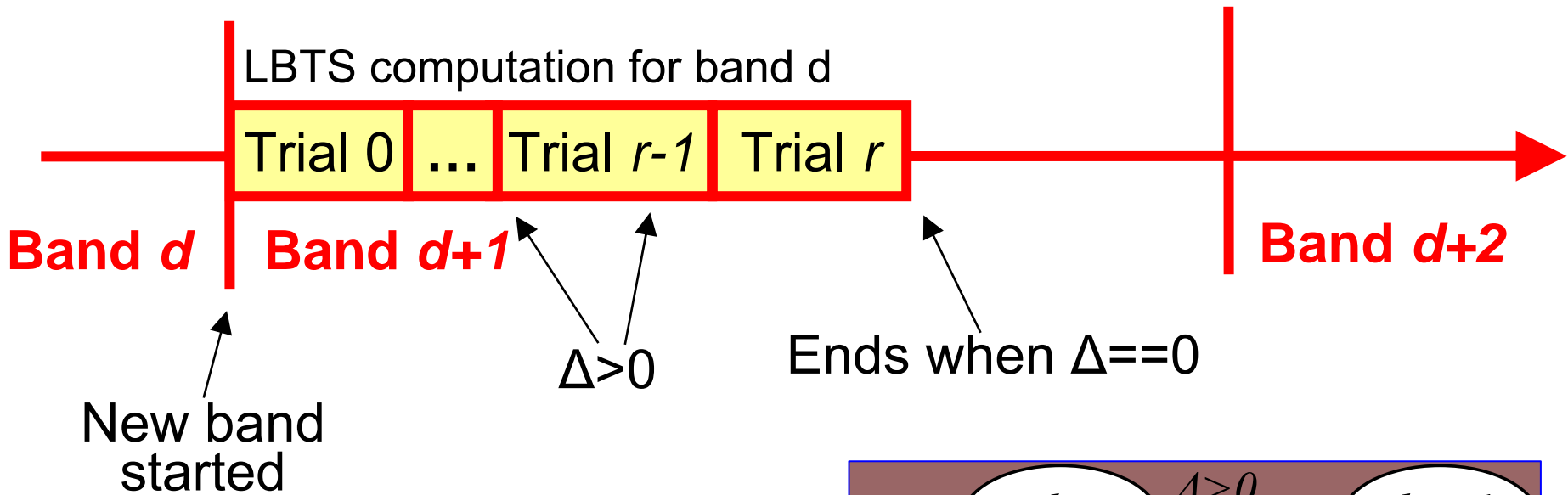
# Basic LBTS Computation Scheme

- For each band  $d$  LBTS computed as sequence of reduction trials
- All TM messages and TSO events tagged with band and trial ID.
- Each trial reduces all  $\langle \tau_i, \delta_i \rangle$  to  $\langle \tau, \Delta \rangle$

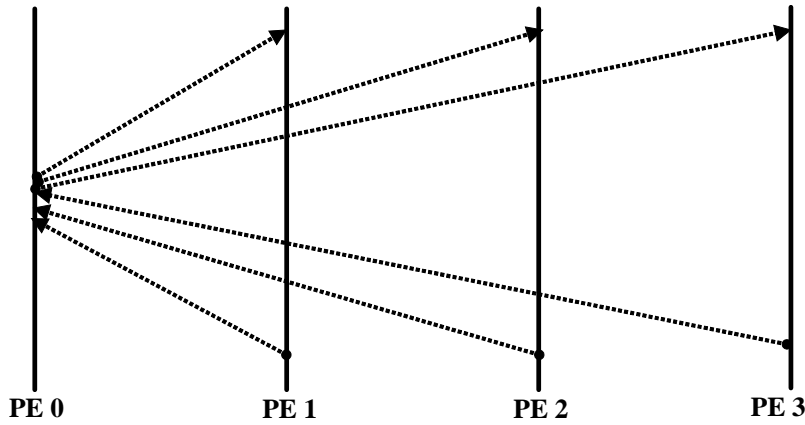
LBTS= $\tau$  when  $\Delta==0$  for some trial  $r$ !



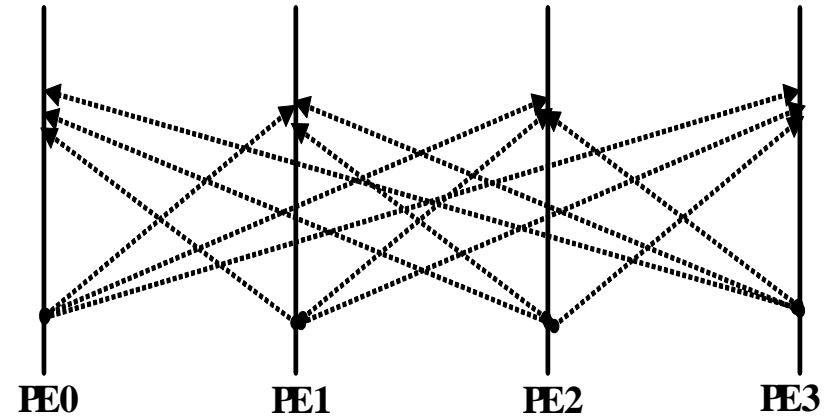
# LBTS Algorithm



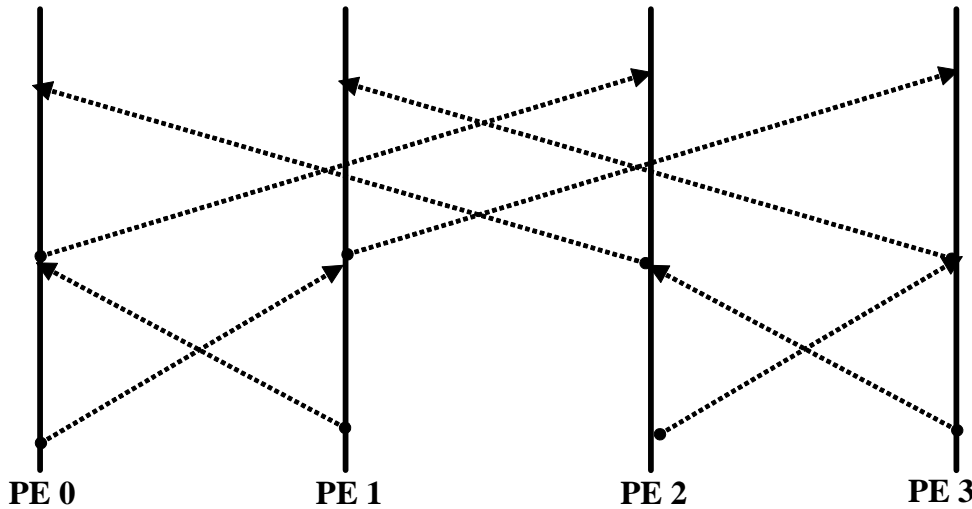
# Asynchronous Distributed Reductions – Patterns



Star pattern



All-to-All pattern

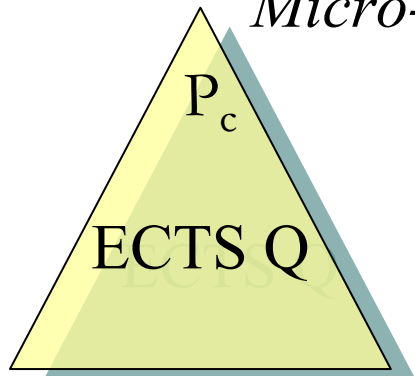


Butterfly pattern

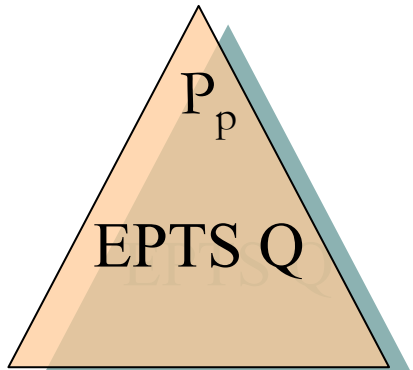
*And combinations of these patterns – at different network levels*

# Micro-Kernel Implementation

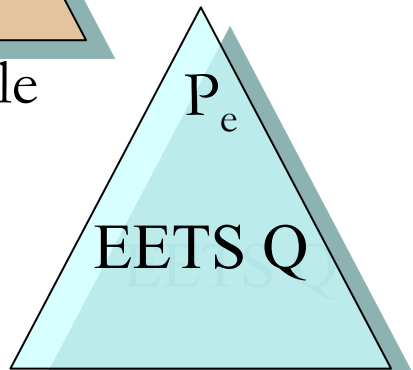
*Micro-Kernel*



Committable

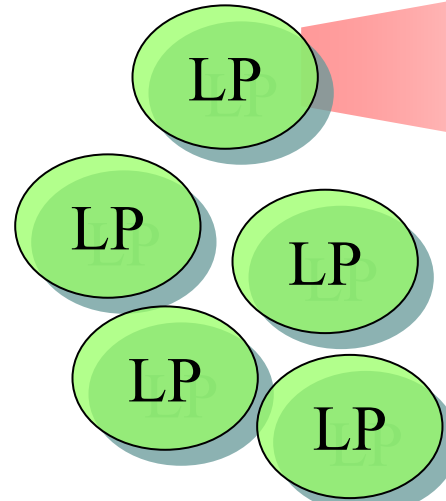


Processable

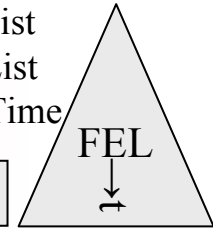


Emittable

*User LPs*

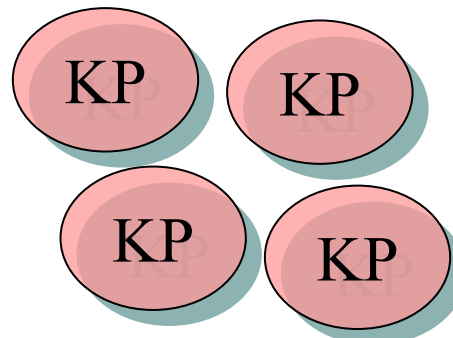


Future Event List  
Proc'd Event List  
Local Virtual Time



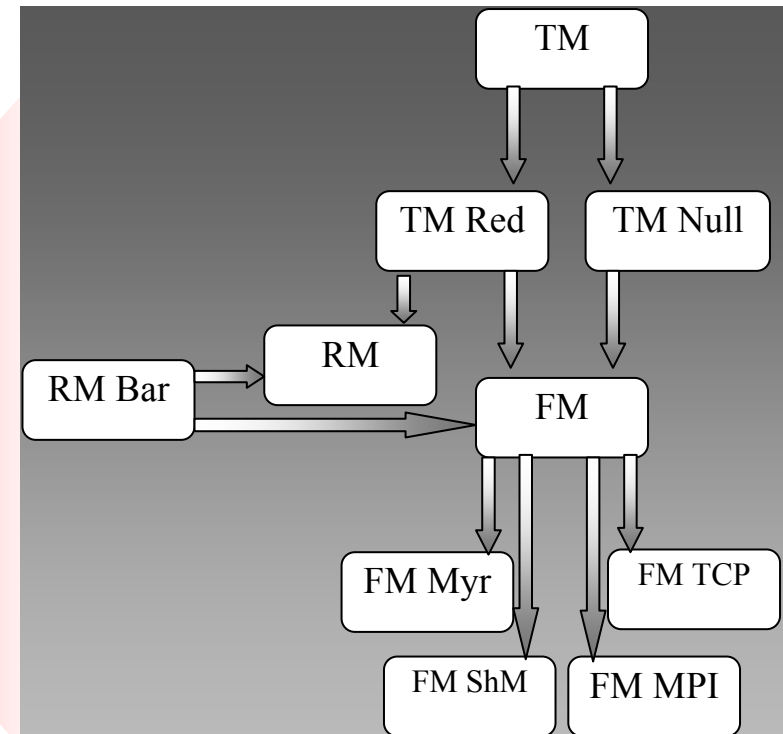
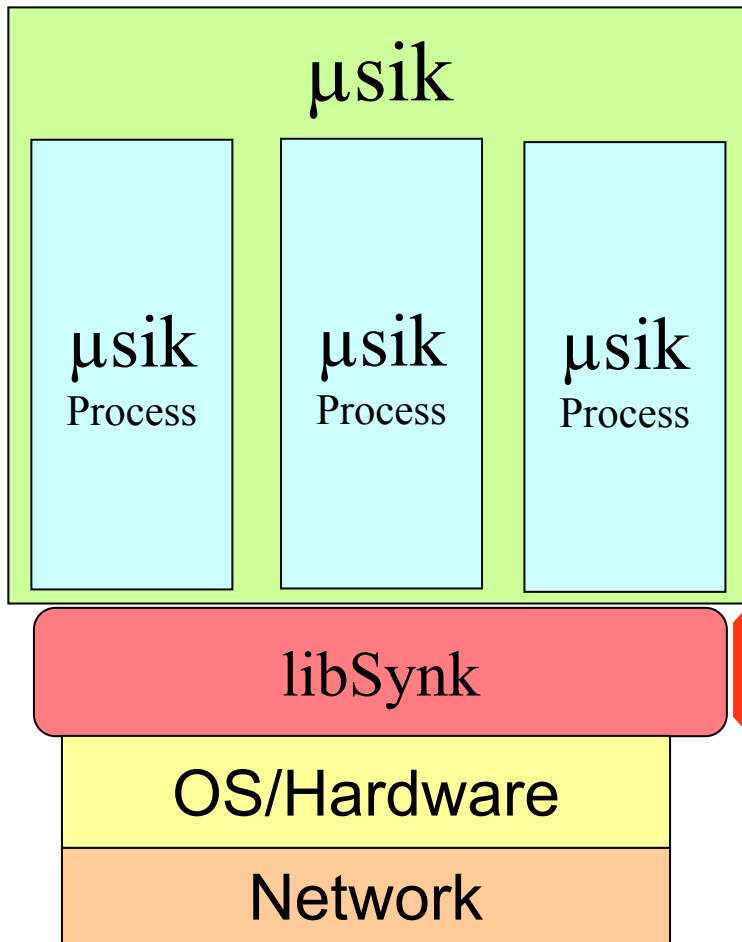
When update kernel Q's?

- New LP added or deleted
- LP executes an event
- LP receives an event



*Kernel LPs*

# Underlying Engine Software Architecture



X → Y Implies X uses Y

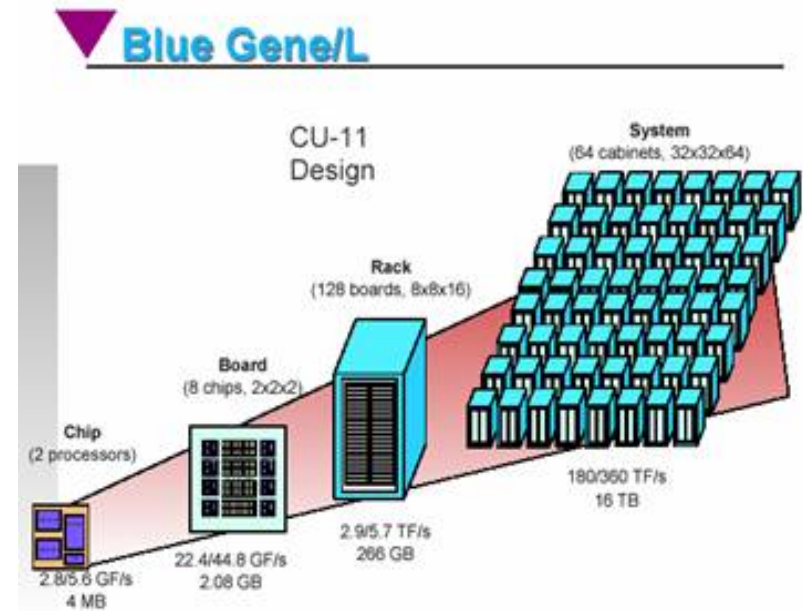
- PHOLD Benchmark (Jugglers)
- Results on IBM Blue Gene

*“Scaling Time Warp-based Discrete Event Execution to  $10^4$  Processors on the Blue Gene,” Proceedings of the ACM Computing Frontiers, May 2007, Ischia, Italy*

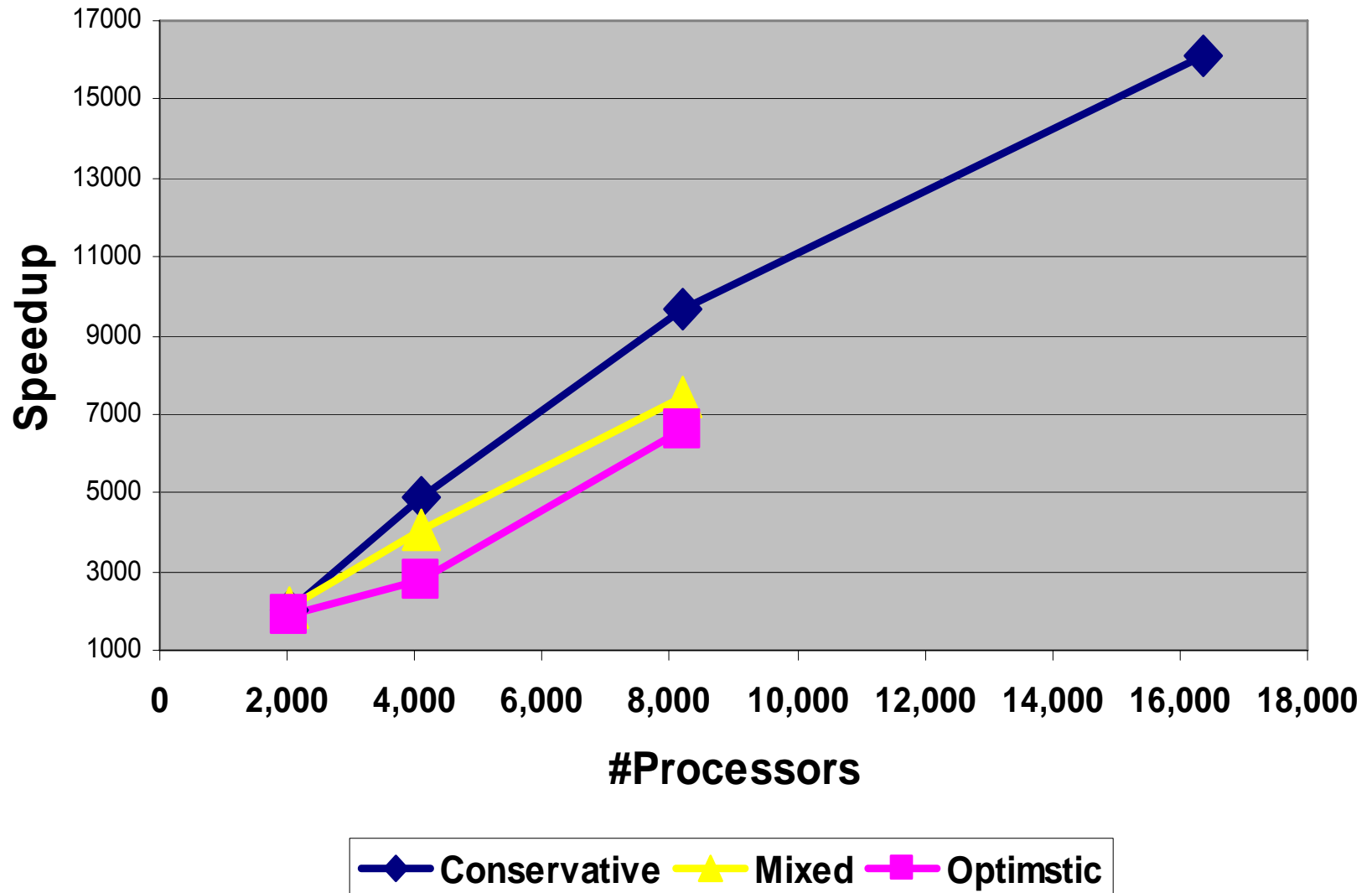
- Relatively fine grained
  - ~5 microseconds computation per event
- Conservative
  - `LPi.enable_undo( false )`
- Optimistic
  - `LPi.enable_undo( true, RA=10*LA )`
  - Reverse-computation
  - Reversible random number generator
- Mixed Mode
  - Even numbered LPs are conservative
  - Odd numbered LPs are optimistic

# Benchmark Runs on Blue Gene Watson

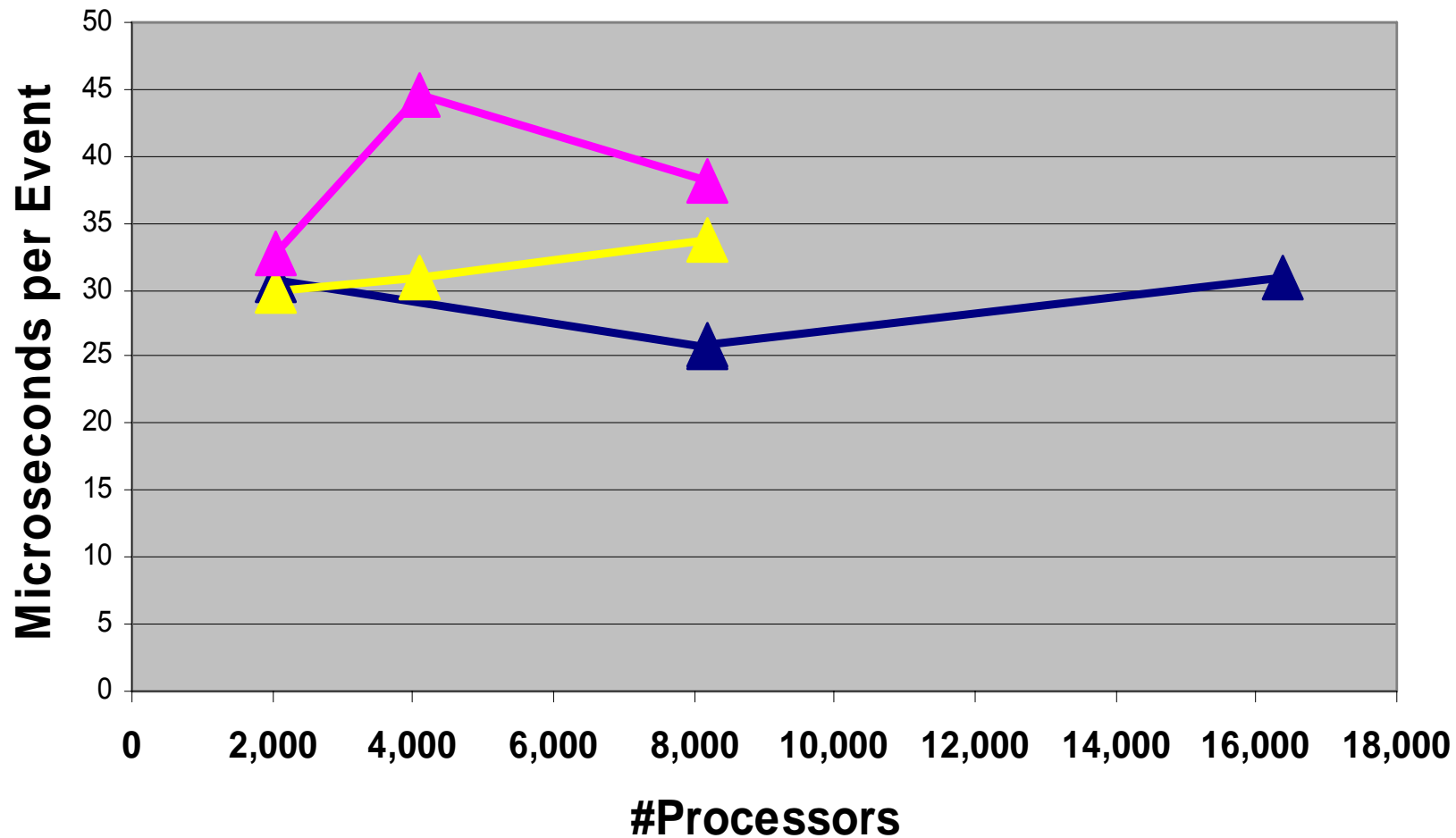
- At IBM T.J.Watson Research Center, New York (Ranked 2<sup>nd</sup> in Top 500)
- 16 racks, 1K nodes/rack, 2 cores/node = 32K cores total
- We gratefully acknowledge BGW Days Program from IBM



# μsik Speedup

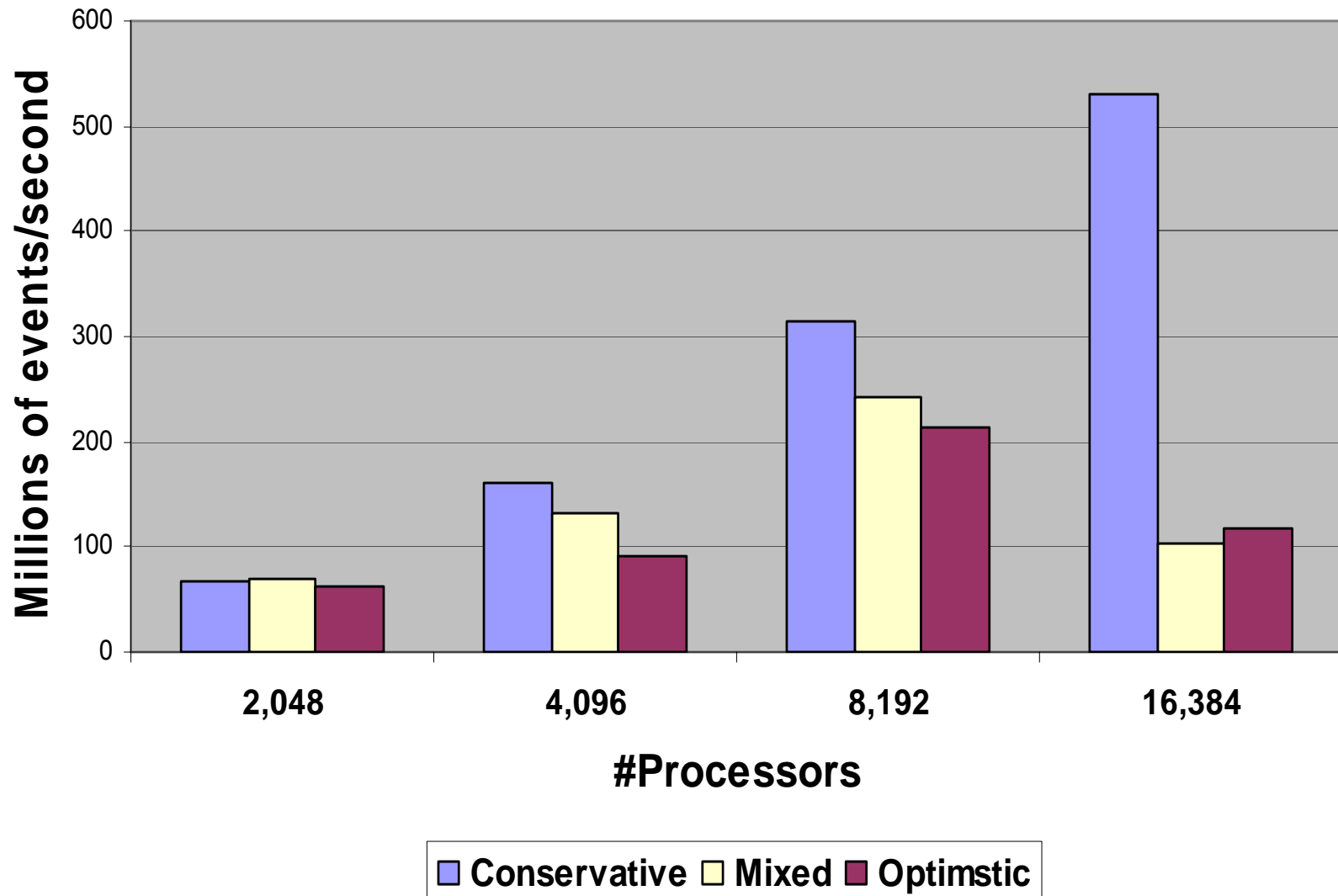


# $\mu$ sik Event Cost

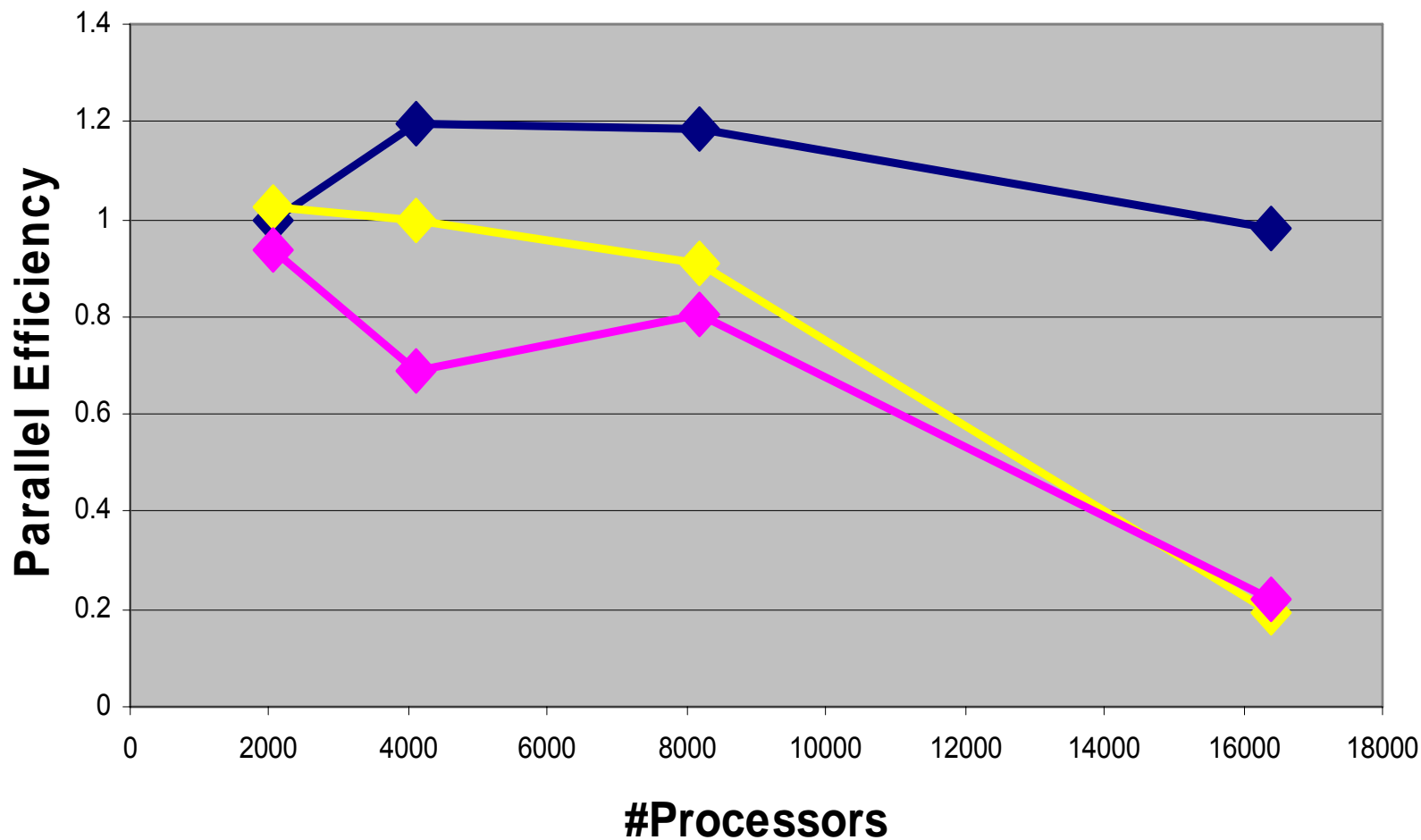


▲ Conservative ▲ Mixed ▲ Optimstic

# $\mu$ sik Event Rate



# $\mu$ sik Parallel Efficiency

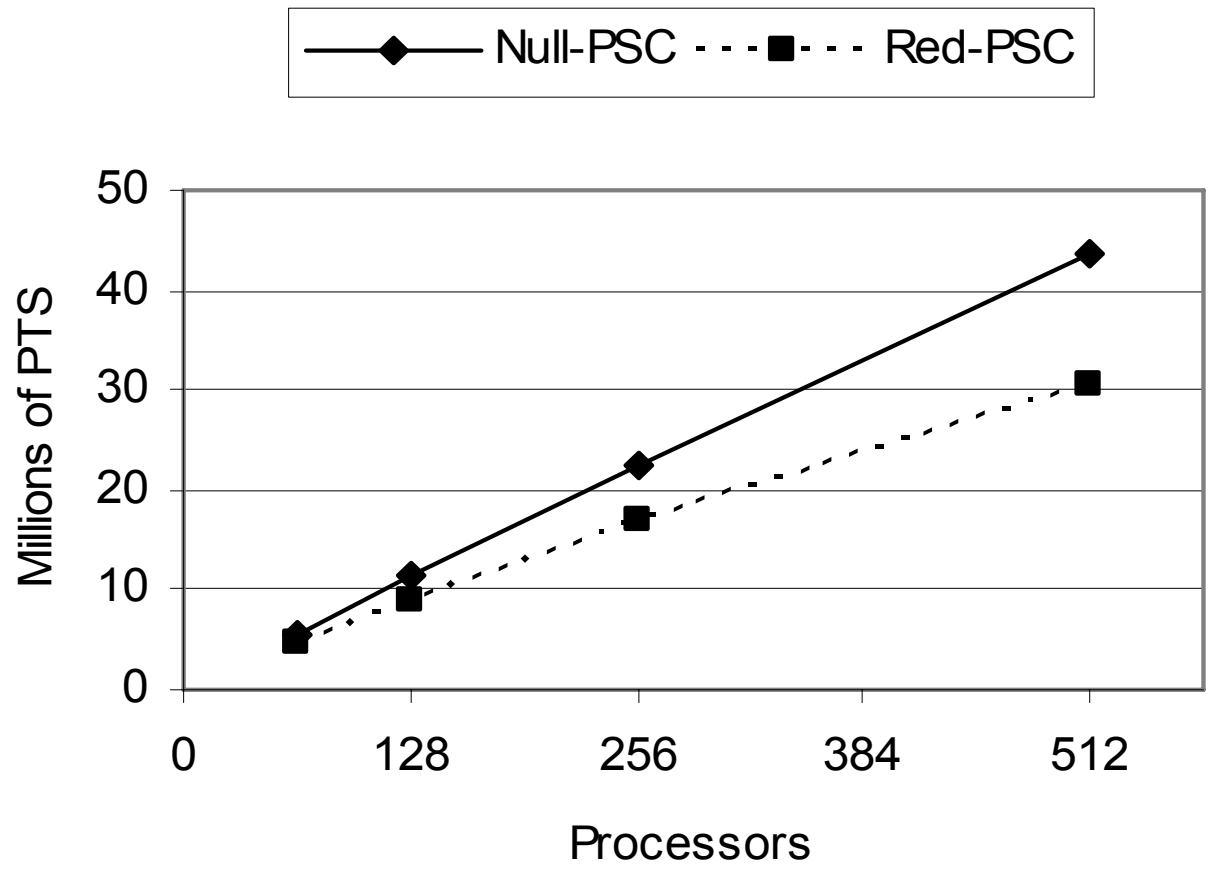


◆ Conservative ◆ Mixed ◆ Optimistic

# Null Message-based Synchronization

- Can improve upon reductions-based global synchronization
- Null messages depend on topology, and exploit local interactions, avoiding global operations

# Null Message Performance on the Lemieux Supercomputer

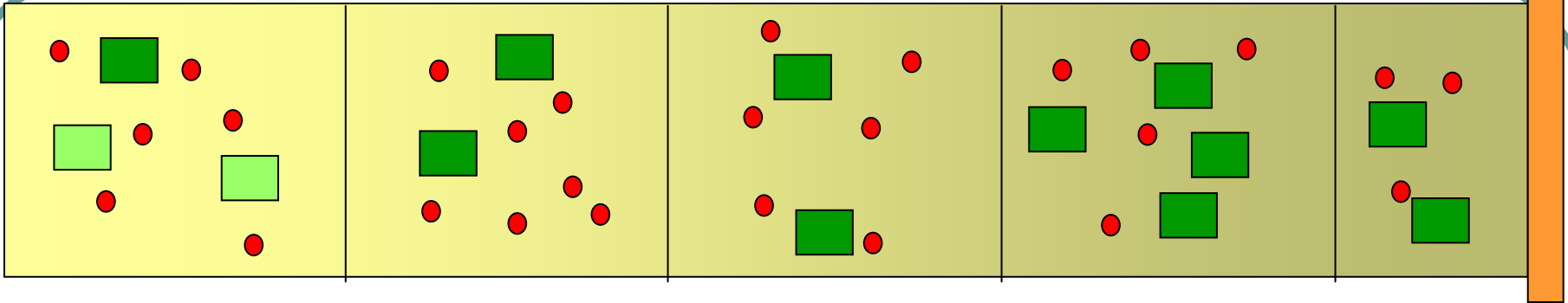


Run times (sec)

CPUs	Null	Red
64	420	508
128	414	523
256	420	563
512	436	620

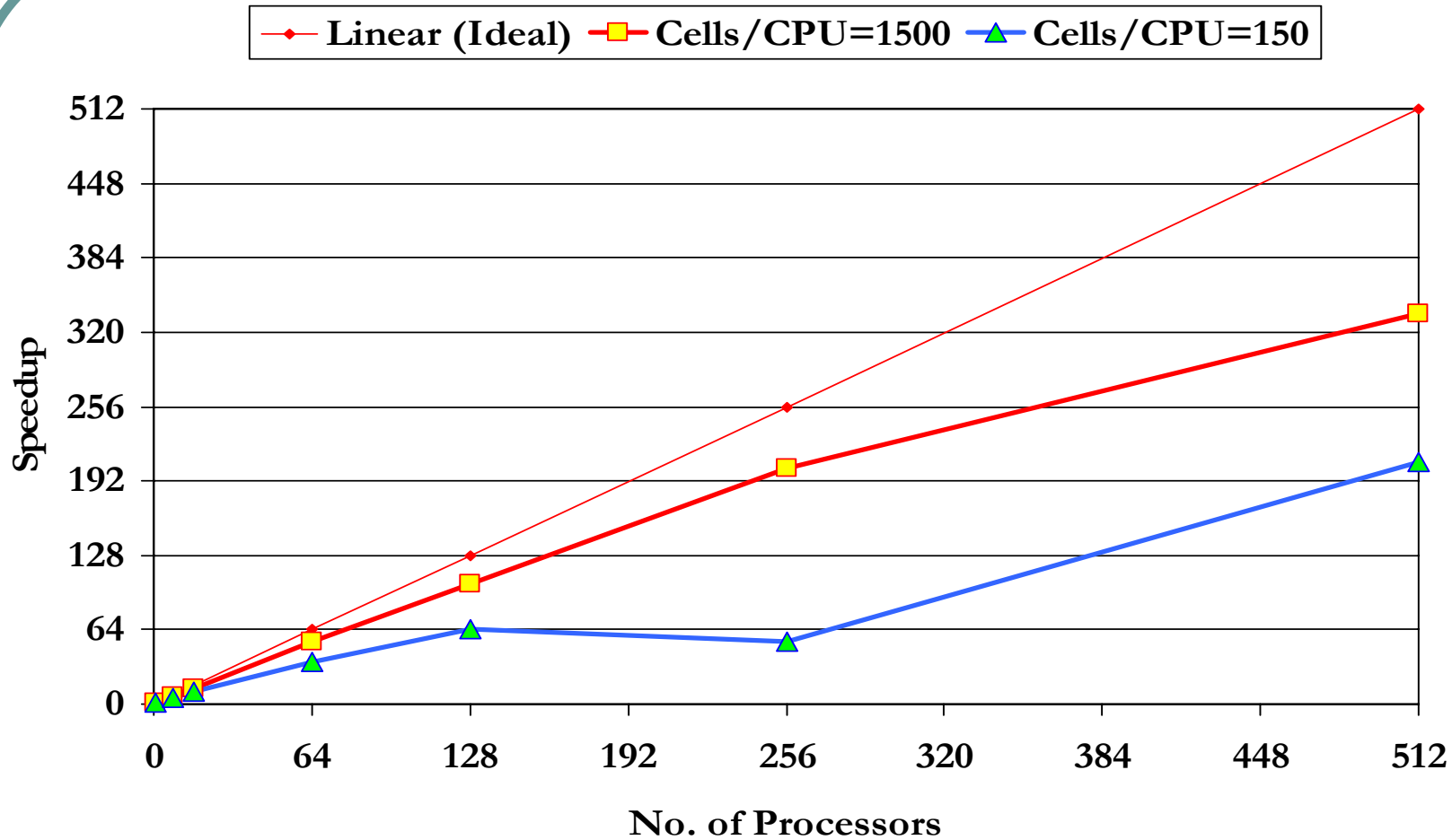
- See ASYM'07
  - [www.pads07.org/async](http://www.pads07.org/async)

# Hybrid Particle-In-Cell Model



- Field update notification events
- Particle transfer events
- Advance cell events

# 1-D Hybrid Shock DES Model



- Problem size scaled with processors: Cells/CPU = 150, 1500; Ions/Cell = 100
- Largest: 76.8 million ions (100 ions/cell x 1500 cells/CPU x 512 CPUs)

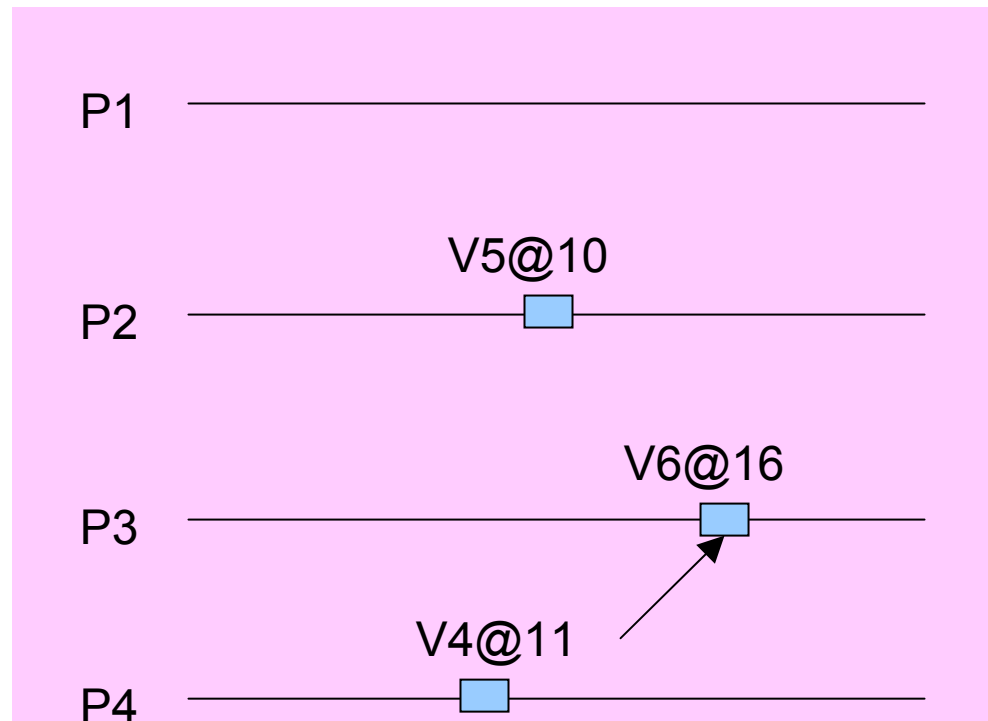
# Alternative Modeling

- Lookback
- Approximate Time

- When lookahead is hard to define or extract, see if the model is “resilient”
- Lookback is an LP’s ability to “tolerate” a temporal error of up to LB
- Toleration is typically via fix up computation
- This is analogous to local rollback.
- E.g., an LP with a lookback of LB can “look back” in the past from  $T$  to  $T-LB$  with guaranteed accuracy of fix up.

# Approximate Time

- Event timestamps are only a best guess
- There is often ambiguity in precision
- E.g., vehicle traveling  $5 \pm 2$  seconds?
- Can we exploit this modeling ambiguity for concurrency?



## Approximate Time (continued)

- Approximate Time (AT) can help relieve tight coupling due to timestamps *dynamically*
- Simulation engine can choose timestamp for efficiency
- Possible to extract concurrency dynamically, even though basic lookahead is zero
  - *i.e.,  $0 \pm a$  can help uncover lookahead of up to  $a$*

- HLA
- XMSF

# Massively Multiplayer Gaming

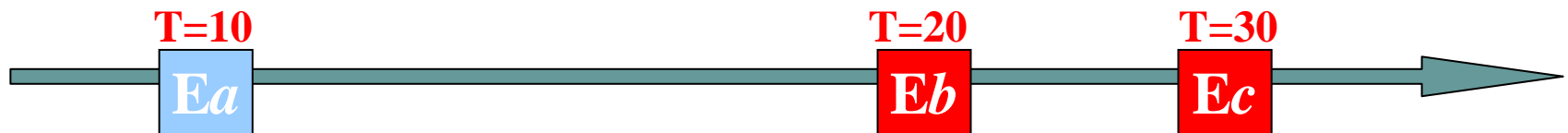
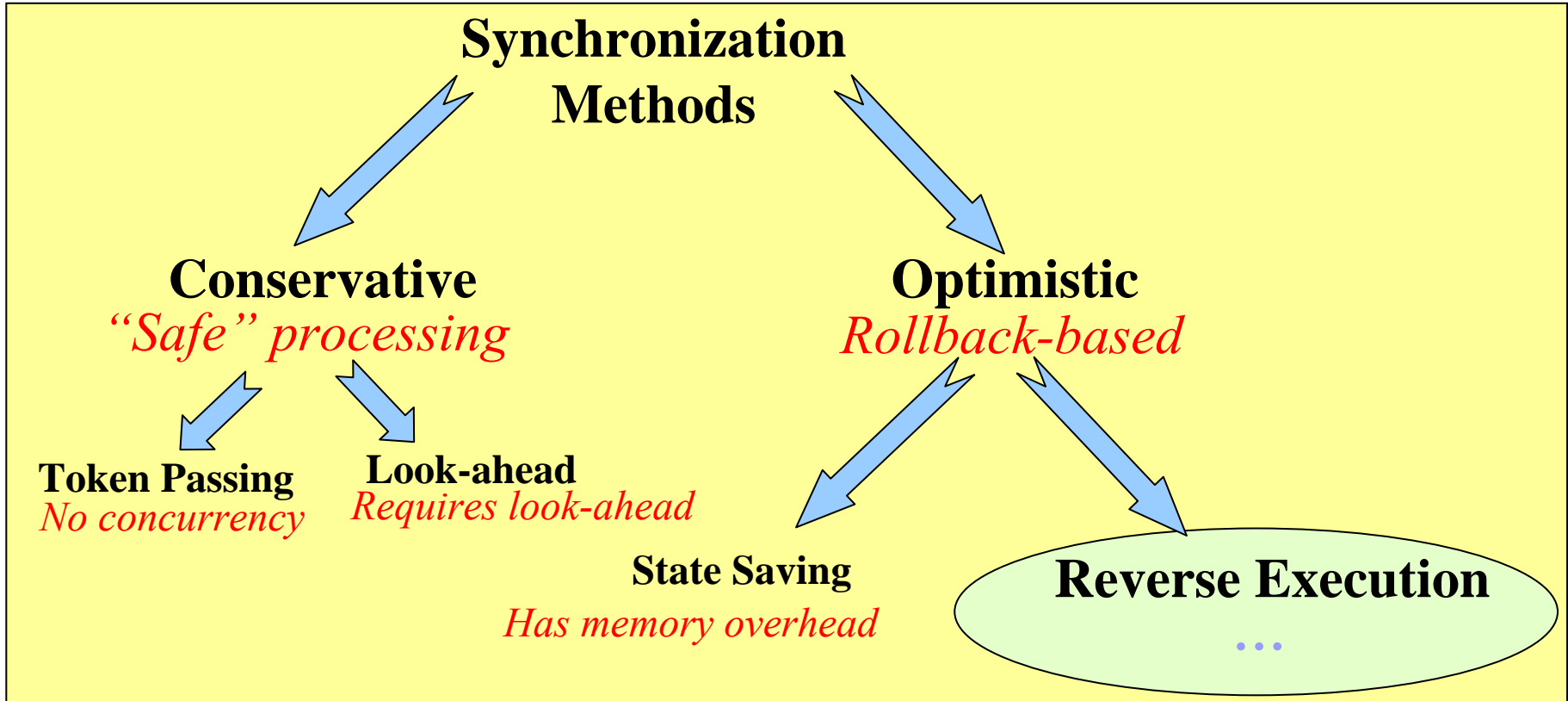
# Alternative Synchronization

- Critical Channel Traversal
- Lookahead Extraction
- Mixed Mode (Optimistic + Conservative)
- Reverse Computation

# Reverse Computation

# Parallel Execution Techniques

**Goal:** Ensure global timestamp-ordered processing.  
=> Synchronization among simulators required.



## Problem: To support rollback for optimistic simulation

- Traditional Approach

- *State saving*

- Undo by saving and restoring e.g.  
 $\{\text{save}(x); x=x+1\}$   
 $\{\text{restore}(x)\}$  →

- Disadvantages ←

- Large state memory size
- Memory copying overheads
- Poor match for large-scale, fine-grained applications.

- New Alternative

- *Reverse Computation*

- Undo by executing in reverse e.g.  
 $\{x=x+1\}$   
 $\{x=x-1\}$  →

- Advantages ←

- Reduced state memory size
- Reduced overheads; moved from forward to reverse
- Excellent match for large, high-performance simulations
- Can be automated.

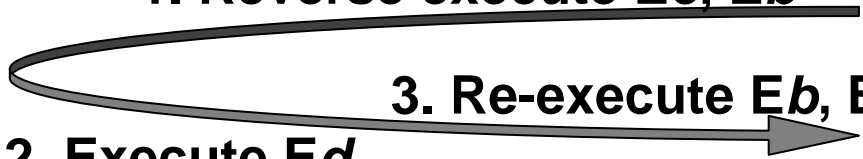
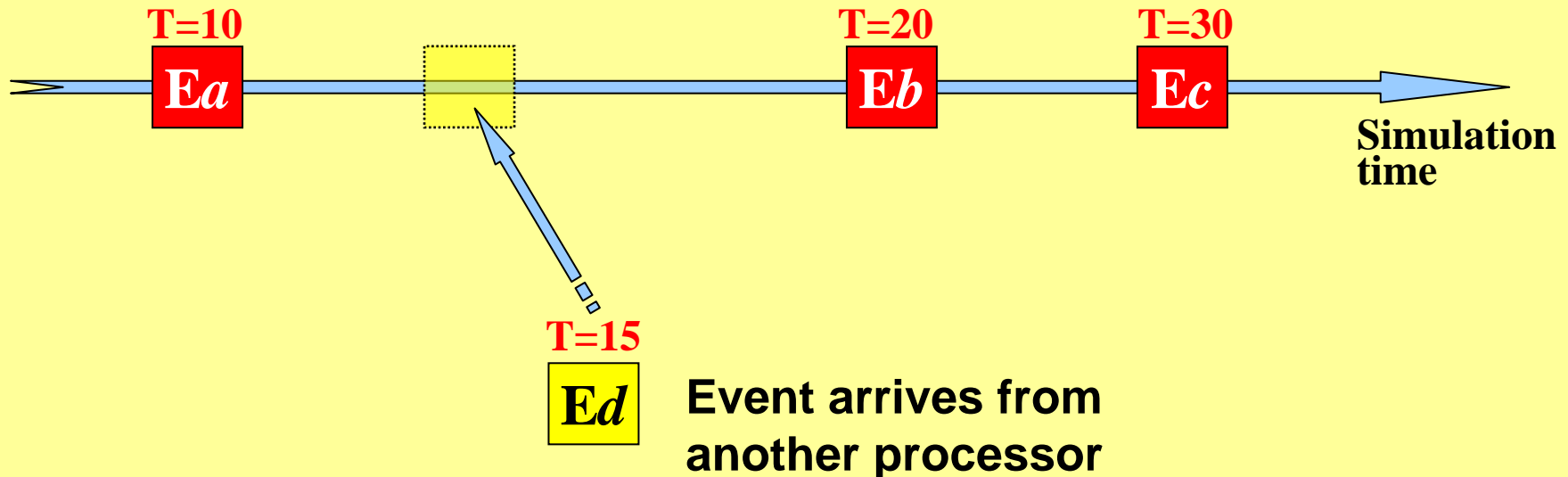
# Rollback Using Reverse Execution

*Idea: Execute inverse operations to undo forward computation.*

Example: Events  $E_a, E_b, E_c$  are processed. Later  $E_d$  arrives.

**Rollback:**

1. Reverse execute  $E_c, E_b$
2. Execute  $E_d$
3. Re-execute  $E_b, E_c$

A diagram illustrating the rollback process. It shows a horizontal timeline with three red boxes labeled 'Ea', 'Eb', and 'Ec' at time steps T=10, T=20, and T=30 respectively. A blue arrow labeled 'Simulation time' points to the right. A yellow box above the timeline contains the text 'Rollback:' followed by three numbered steps: '1. Reverse execute Ec, Eb', '2. Execute Ed', and '3. Re-execute Eb, Ec'. A curved arrow starts from the right side of step 1 and points back to the left side of step 2. A straight arrow starts from the right side of step 2 and points to the right side of step 3.

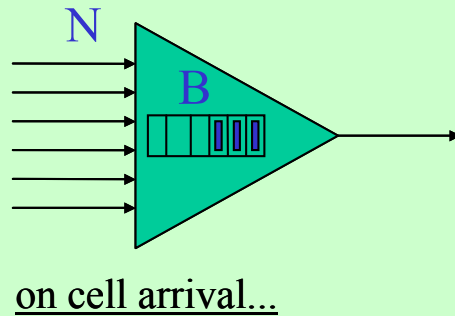
# Why Choose Reverse Execution?

- **Advantages:**
- Dynamic concurrency extraction
  - Due to optimistic processing
- Low memory overhead
  - No need to save state snapshots
- Can be automated
  - Pre-processor for production use

# RC Example: ATM Multiplexer

**Example:**  
Queue with  
**N** inputs  
and queue  
size limit **B**

*e.g.* ATM  
Multiplexer



**Original**

```
if( qlen < B )
  qlen++
  delays[qlen]++
else
  lost++
```

**State Size**  
B+2 words

**Forward**

```
if( qlen < B )
  b = 1
  qlen++
  delays[qlen]++
else
  b = 0
  lost++
```

**Reverse**

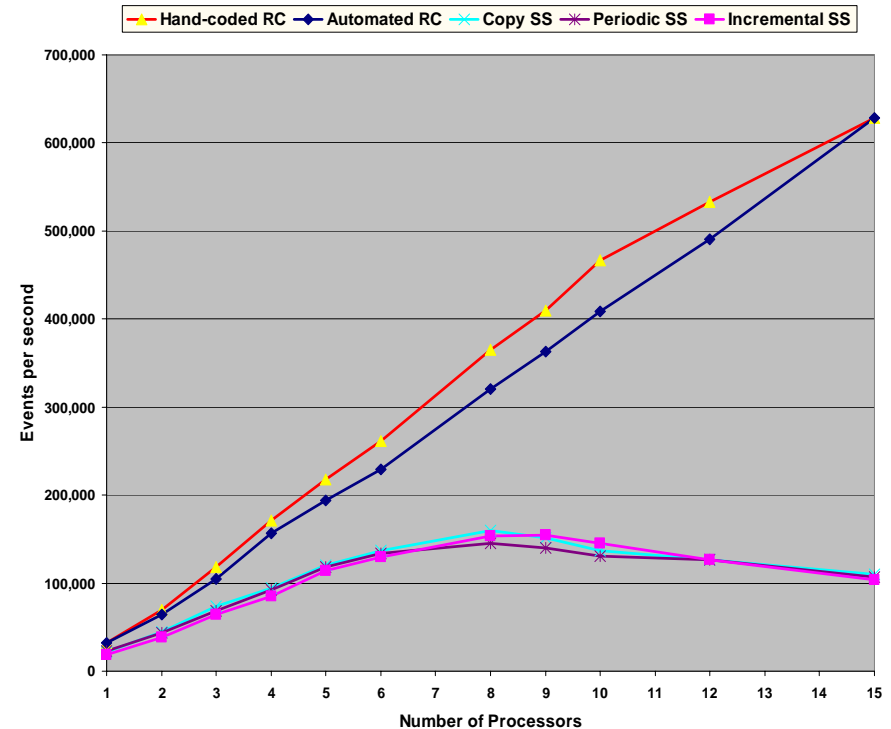
```
if( b == 1 )
  --delays[qlen]
  --qlen
else
  --lost
```

**State Size**  
1 bit

# RC Features & Performance

- Constructive operation => zero state for reversibility (e.g.  $x++$ )
- Destructive operation => state needs to be saved (e.g.  $x=y$ )
- Predominantly **constructive** operations => reduced state size
- **Queueing network models** contain many constructive operations
  - random number generation (reversible RNGs)
  - queue handling (swap, shift, enqueue/dequeue, ...)
  - statistics collection (increment, decrement, ...)

## Performance Gains



# Branching Multiple Simulations via Cloning

The End

Questions & Answers?