

Preferred citation style for this presentation

Waraich, Rashid (2008) Parallel Implementation of DEQSim in Java, *MATSim Workshop 2008*, Castasegna, September 2008.

Parallel Implementation of DEQSim in Java

Rashid Waraich

IVT
ETH
Zurich

September 2008

 Institut für Verkehrsplanung und Transportsysteme
Institute for Transport Planning and Systems

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Outline

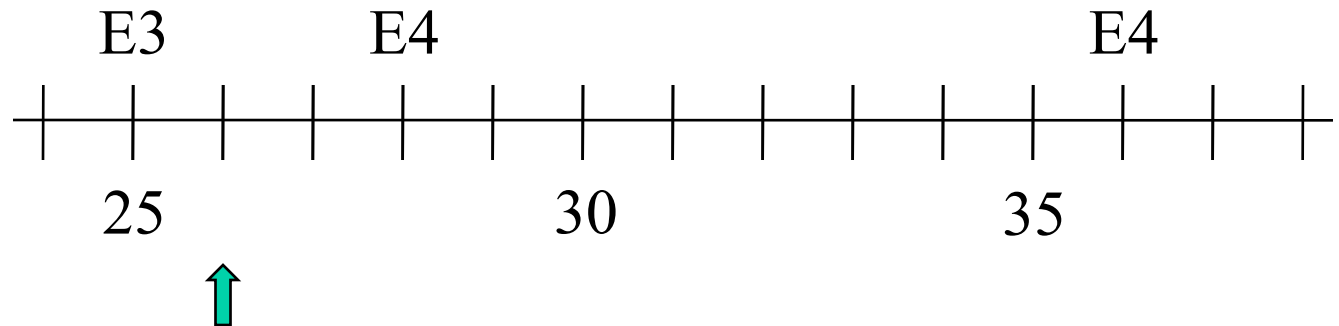
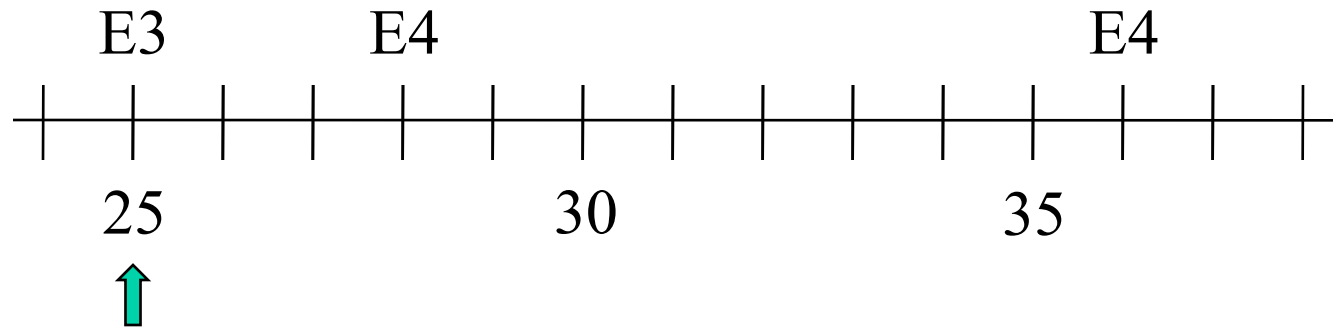
- Introduction
 - Discrete Event Simulation
 - Parallel Discrete Event Simulation
- Java DEQSim Implementation
- Performance Tests
- Future Work / Challenges

Simulation in General

- Continuous time model
 - state variables may change continuously
 - example: change of water temperature, filling up a glass, etc.
- Discrete time model (Discrete Event Simulation - DES)
 - state changes at discrete points in time
 - examples: queueing system, simulation of customers in shopping mall, simulation of air traffic, ...
 - How to advance simulation time?
 - fixed-increment time advance (Java Mobsim)
 - next-event time advance (DEQSim, JDEQSim)

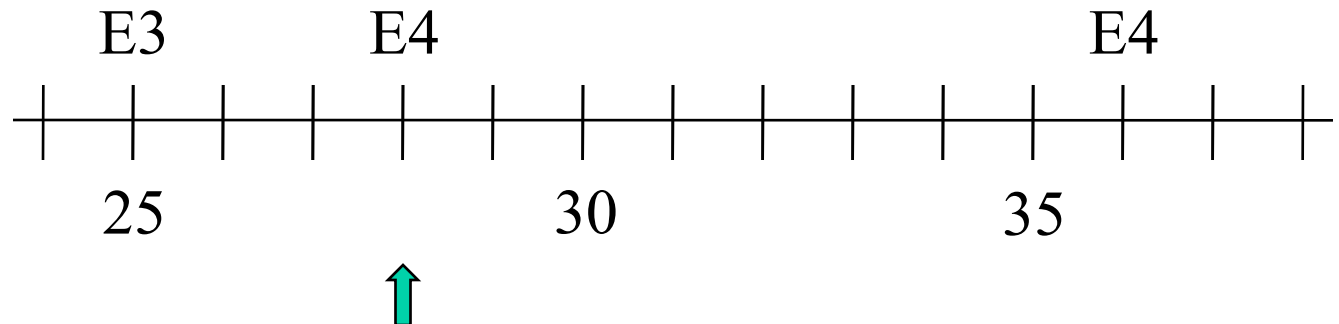
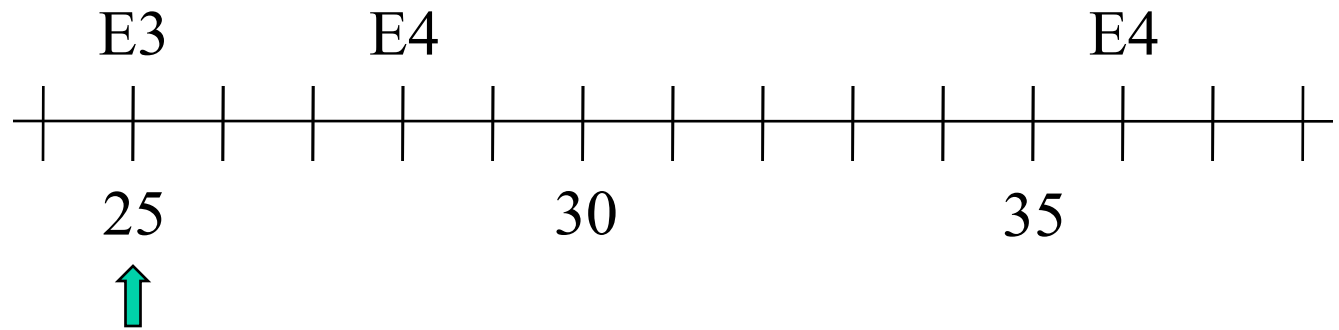
Fixed-increment Time Advance

- Useful, if events occur at fixed length intervals
- Wasteful scanning
- Accuracy problem / tradeoff



Next-event Time Advance

- Time advancement from event to event
- Simulation skips over periods of inactivity
- Called event-driven DES



Parallel Descret Event Simulation (PDES)

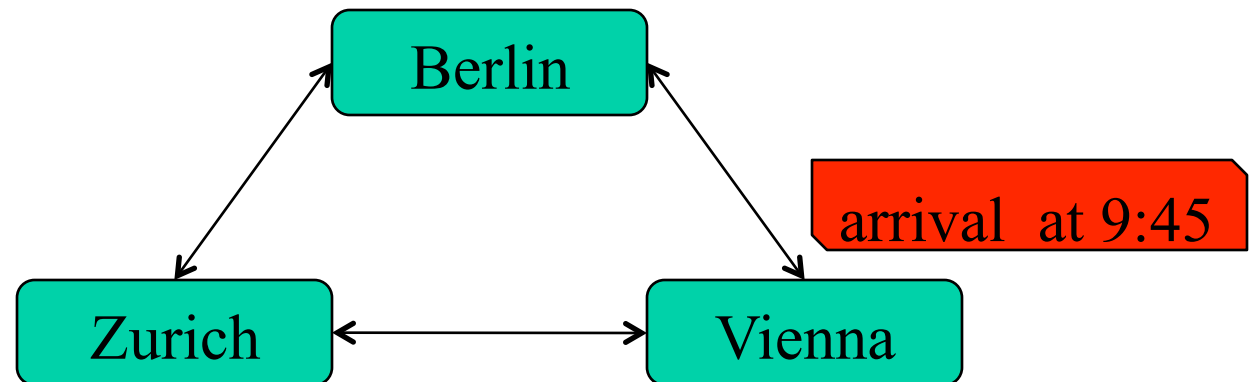
Why PDES?

- DES slow
- slow down factor

How to make simulation parallel?

- partition system into subsystems (logical processes – LP), which can be simulated in parallel

example:



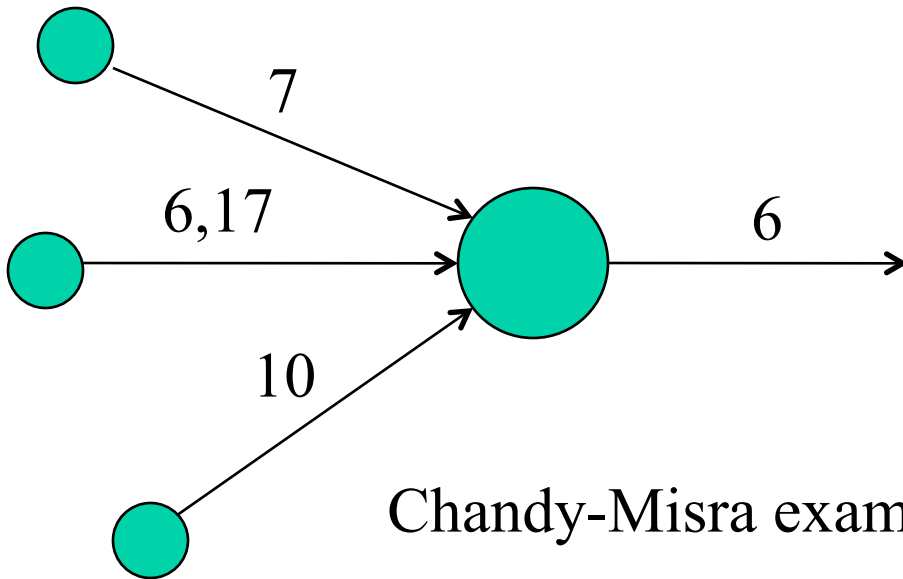
PDES (cont.)

How to preserve causal order of events?

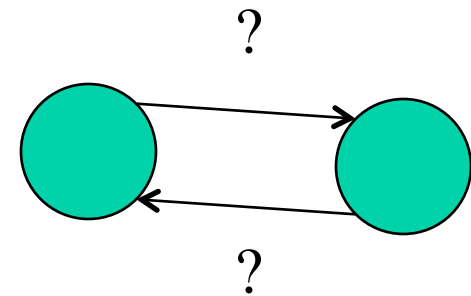
- optimistic algorithms
 - e.g. Time Warp algorithm
- conservative algorithms (DEQSim, Java DEQSim)
 - LP executes safe events only
(e.g. Chandy-Misra algorithm)

Chandy-Misra Algorithm

- Chandy-Misra algorithm
 - null messages, to prevent deadlock
 - problem: lots of null messages
 - need good/large lookahead



Chandy-Misra example



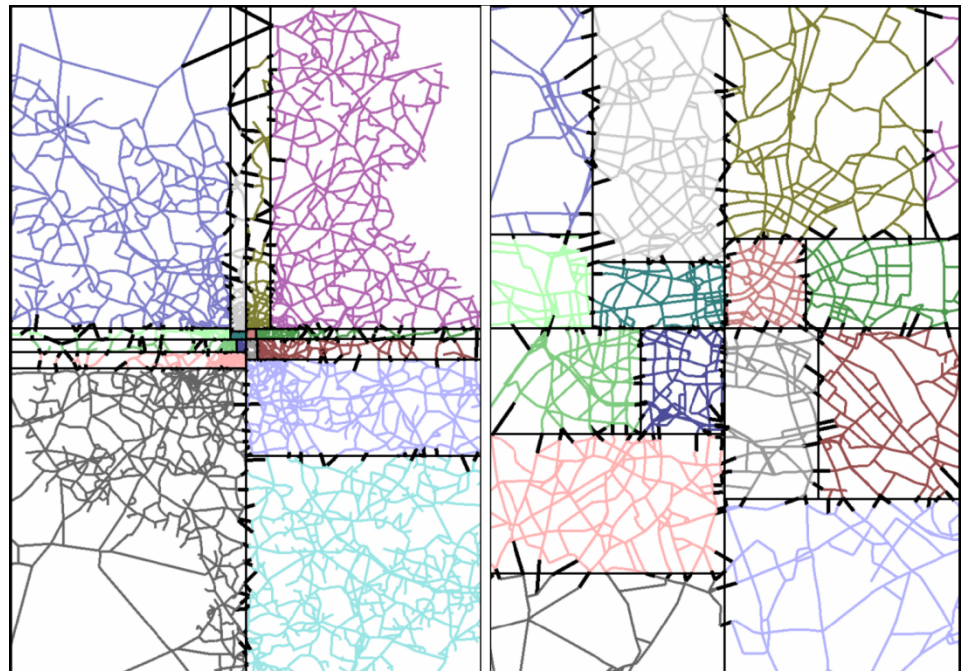
Deadlock

Java DEQSim

- Partition network into Logical Processes (LPs)
- Lookahead
- Synchronization
 - Process Events
 - Synchronization between LPs

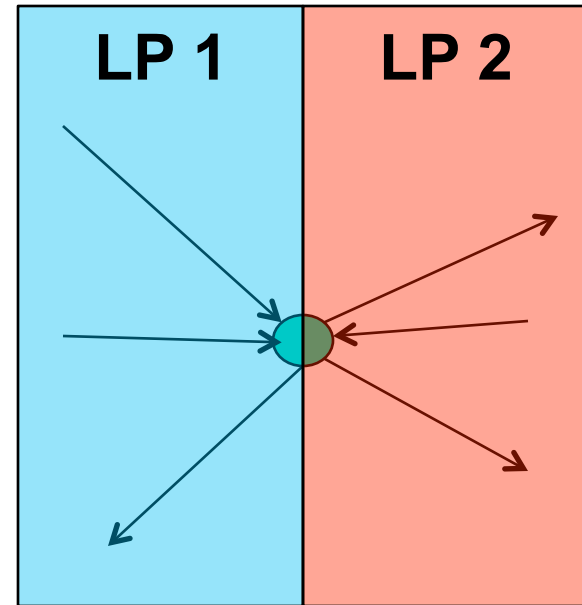
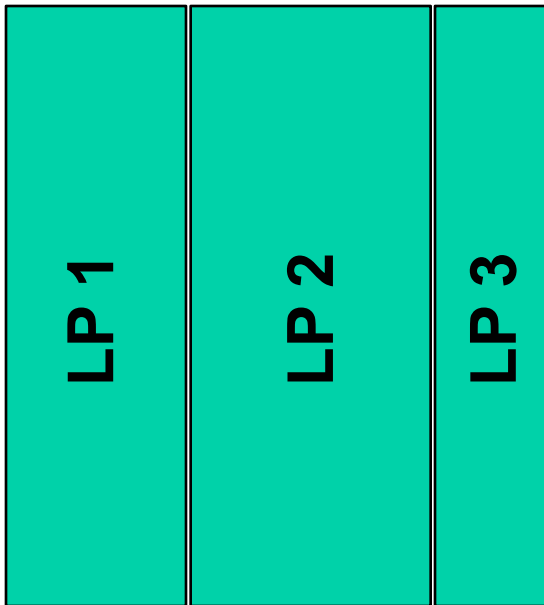
Partitioning (DEQSim)

- Orthogonal recursive bisection (same number of events in each zone)
- Number of zones is power of 2
- Split in middle of roads



Partitioning (Java DEQSim)

- Partition network vertically, each has own queue
- Partition along nodes
- Same number of events per zone
- Less neighbour zones
- Arbitrary number of zones

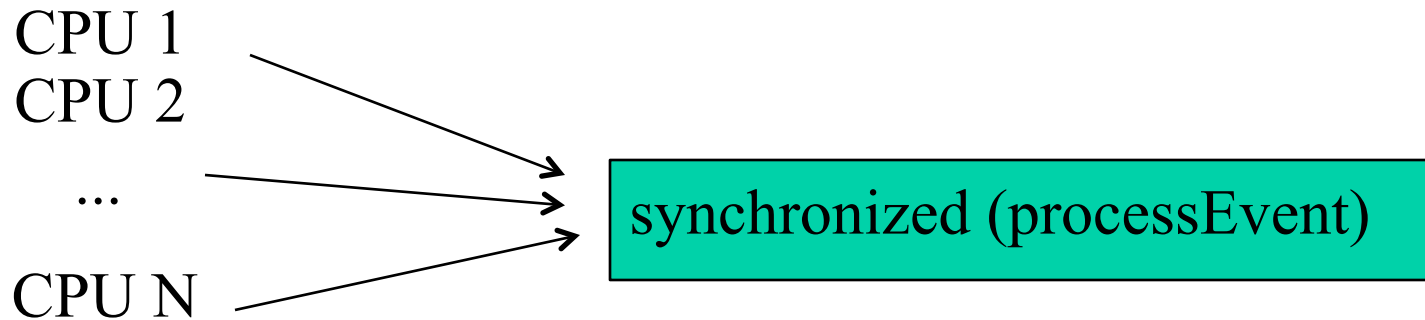


Synchronization / Nullmessages / Lookahead

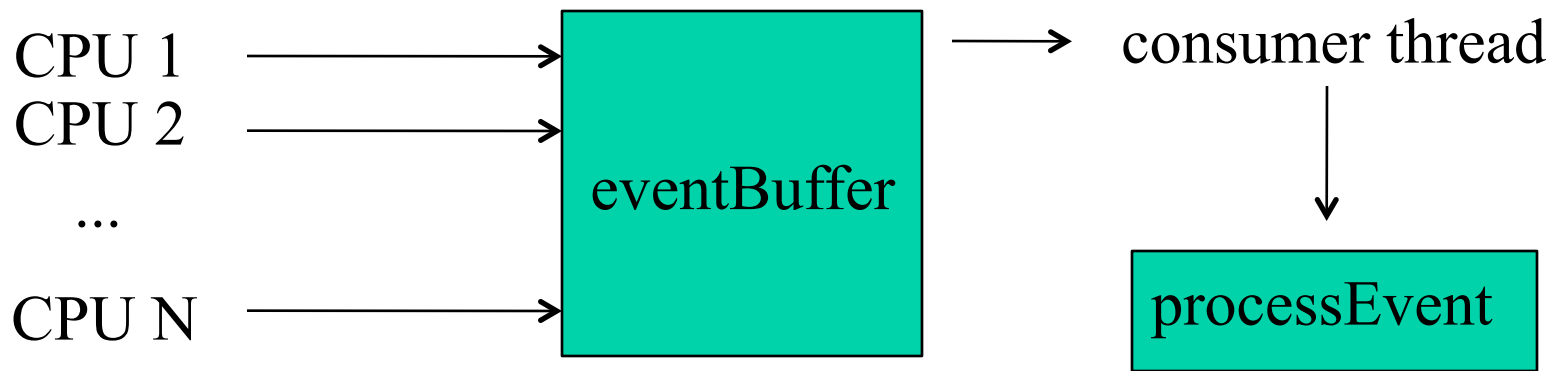
- DEQSim only needs to synchronize at certain predefined points in time
- Java DEQSim: Uses Chandy-Misra algorithm
 - lookahead for reducing number of null messages?
 - plans file knows the future
- Synchronization
 - How handled locking of event process queue
 - How handled locking of queues in each zone

Process Events

initial situation (bottleneck: synchronization)

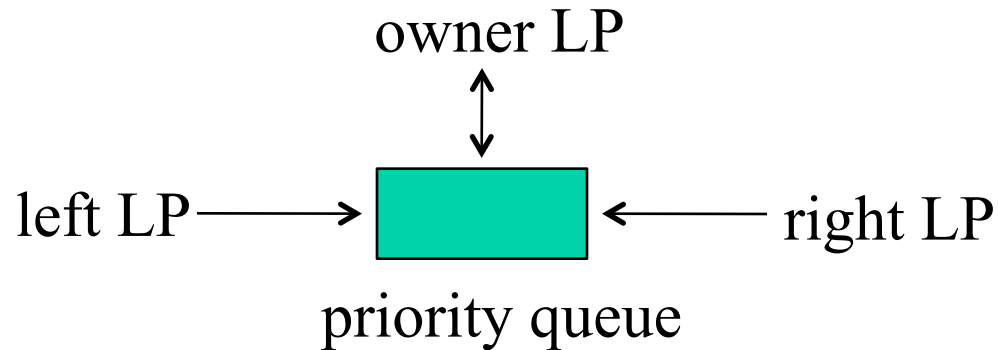


situation now (bottleneck: consumer thread too slow)

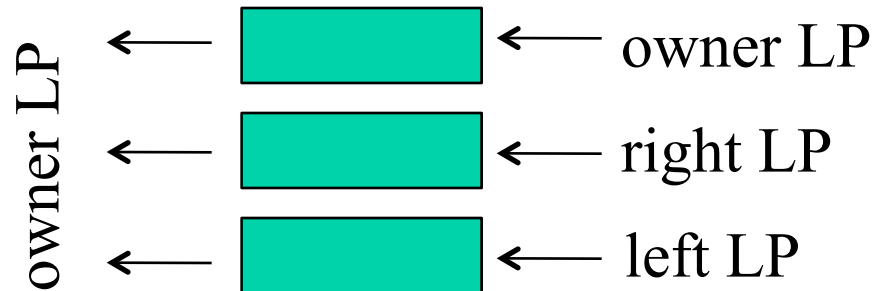


Synchronization between Zones

option 1 (synchronized access on queue)

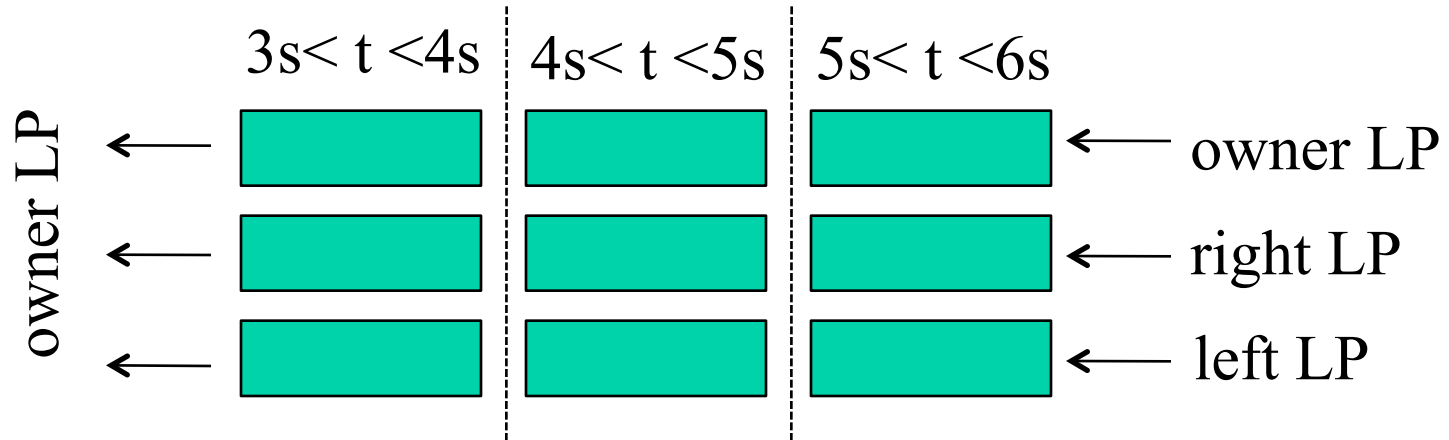


option 2 (lock only queue, which needed)



Synchronization between Zones (cont.)

option 3 (time splitting)

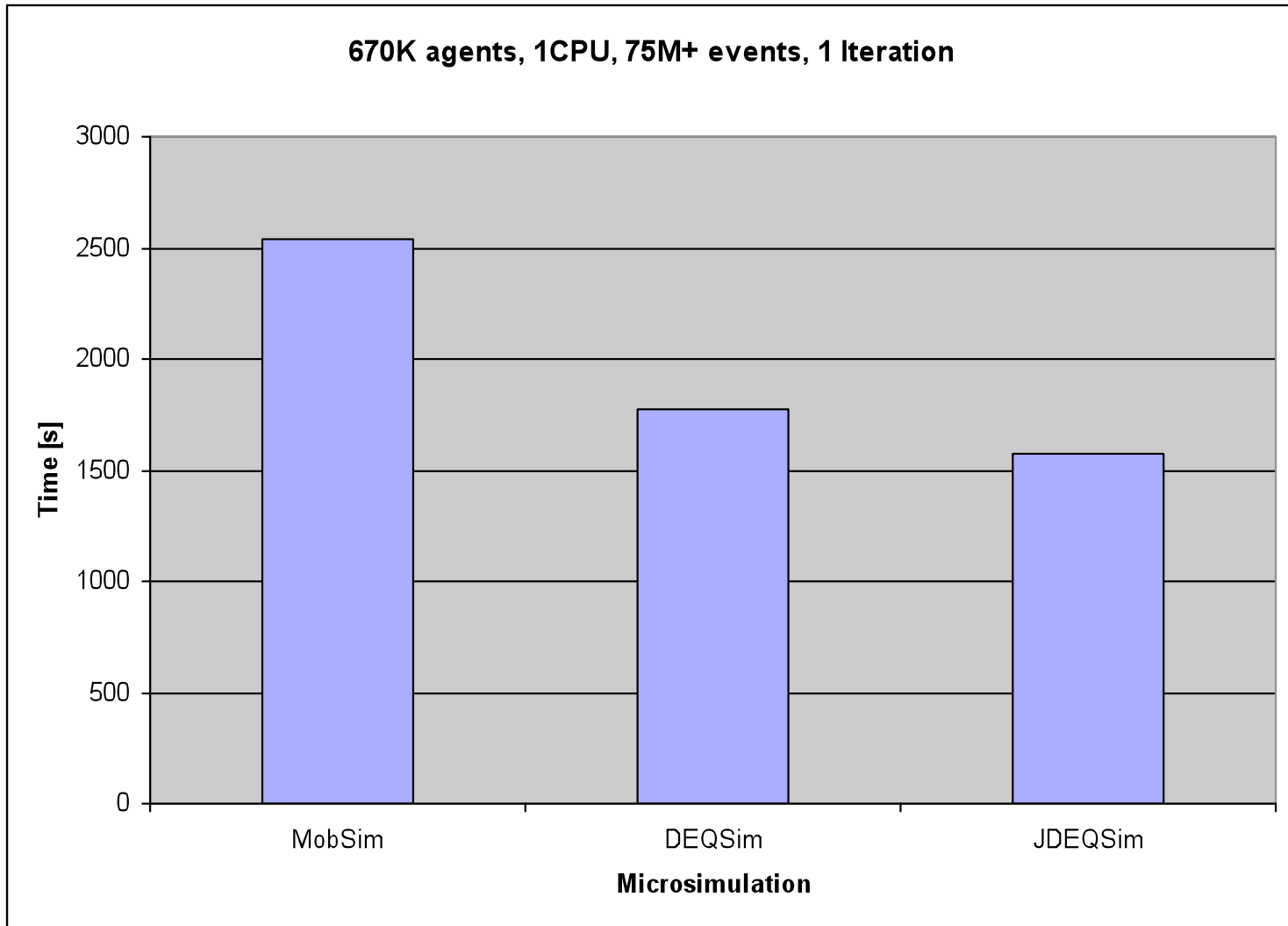


other ideas?

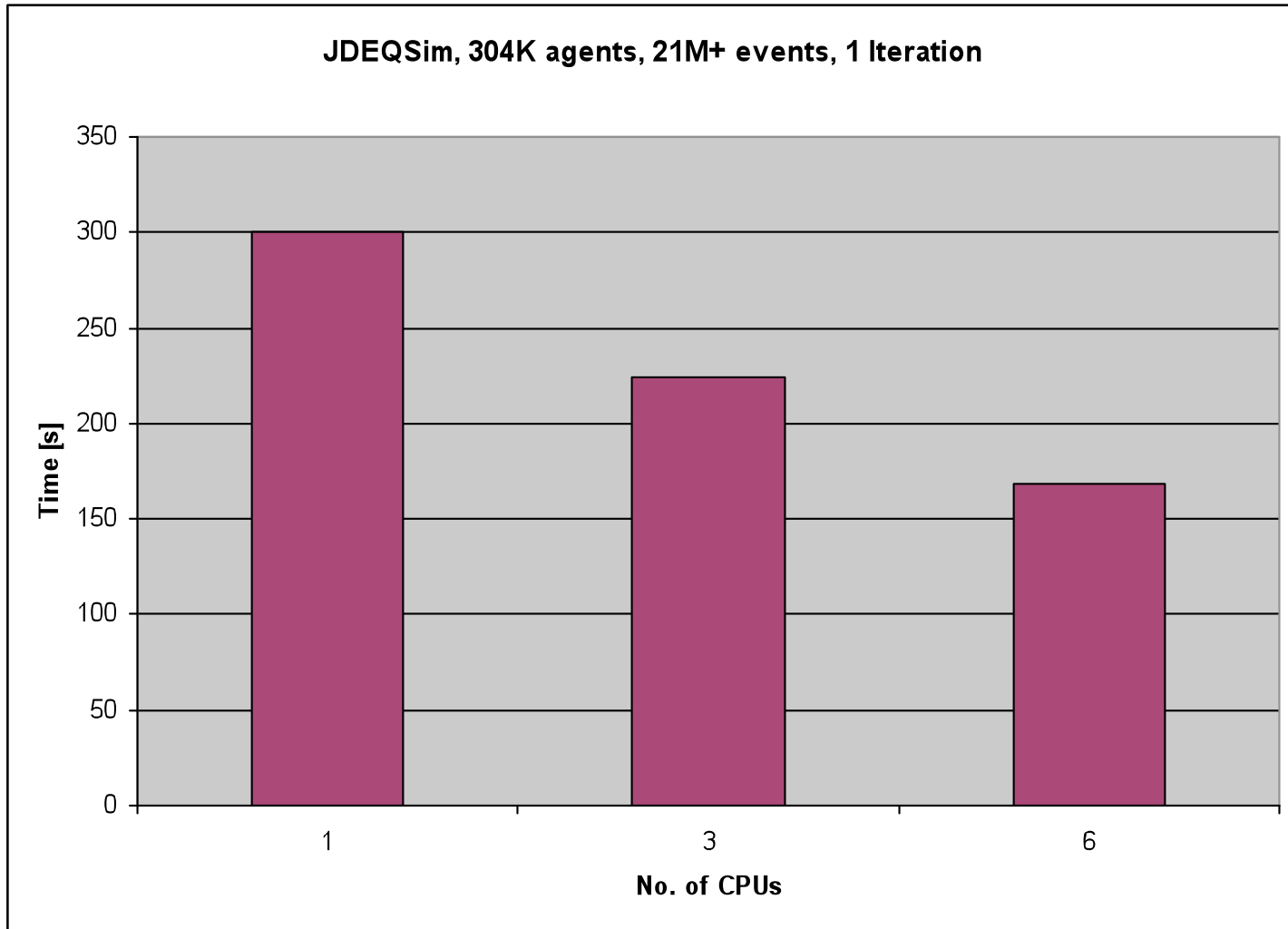
Microsimulation Comparison (for Speed)

	Java MobSim	DEQSim	Java DEQSim
Speed because of programming language	Java	C++	Java
Integrated with rest of MATSim (e.g. no IO-overhead, immediate event handling)	Yes	No	Yes
Support for multithreading	No	Yes	Yes
Advancement of simulation time	Fixed-increment time advance	Next-event time advance	Next-event time advance

Performance Tests I



Performance Tests II



Future Work / Challenges

- **Goal:** One iteration in 15min, approx. 1M links, 7.2M agents, 4.6 trips in average (with approx. 100 links per trip) – currently we would need around 5 hours+ for this on 8 CPUs (with DEQSim)
- How to gain more speed up?
- How to dimension the number of threads for microsimulation and event handling?
- How to do automated „performance regression“ testing?
- Optimistic algorithms?
 - can potentially utilize higher parallelization
 - simpler for the „end user“ to program simulations
 - more difficult to implement than conservative algorithms

How to Gain Speedup?

Java MobSim

simulation + event handling



DEQSim

read + event handling

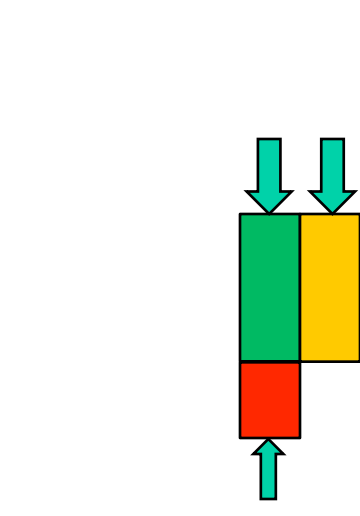


Java DEQSim

simulation



event handling



Future?