On the Relevance of Wait-free Coordination Algorithms in Shared-Memory HPC: The Global Virtual Time Case

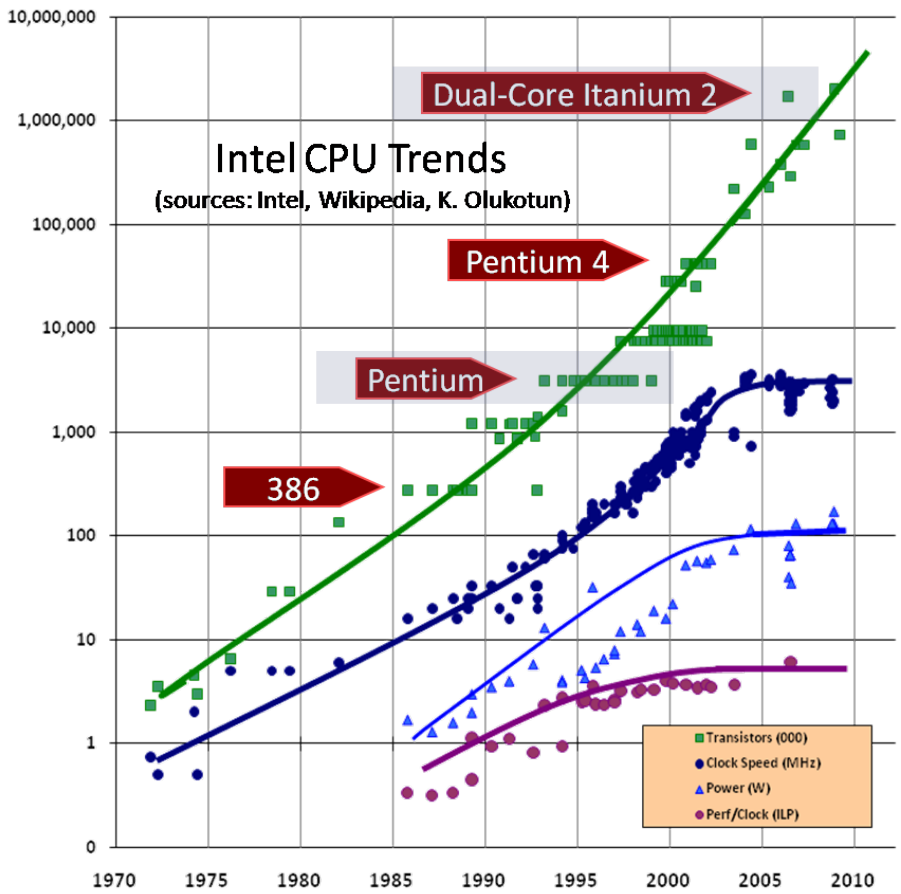
SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Pellegrini
Francesco Quaglia

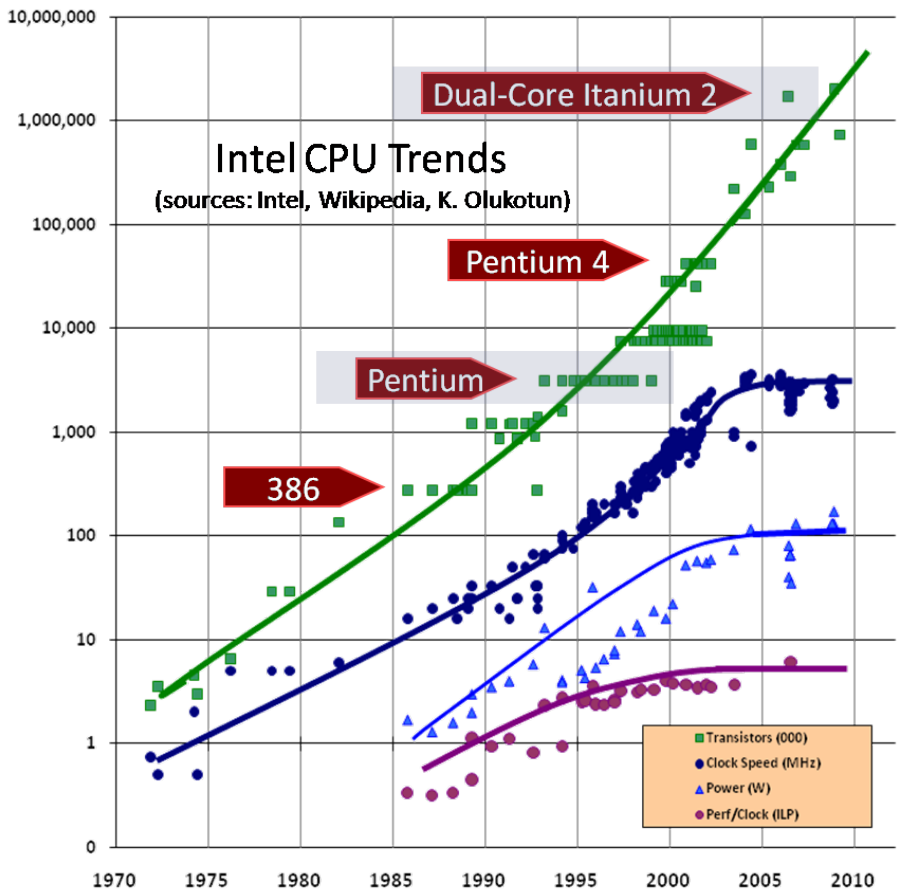
High Performance and Dependable
Computing Systems Group
Sapienza, University of Rome

InfQ 2014

Current Technological Trend

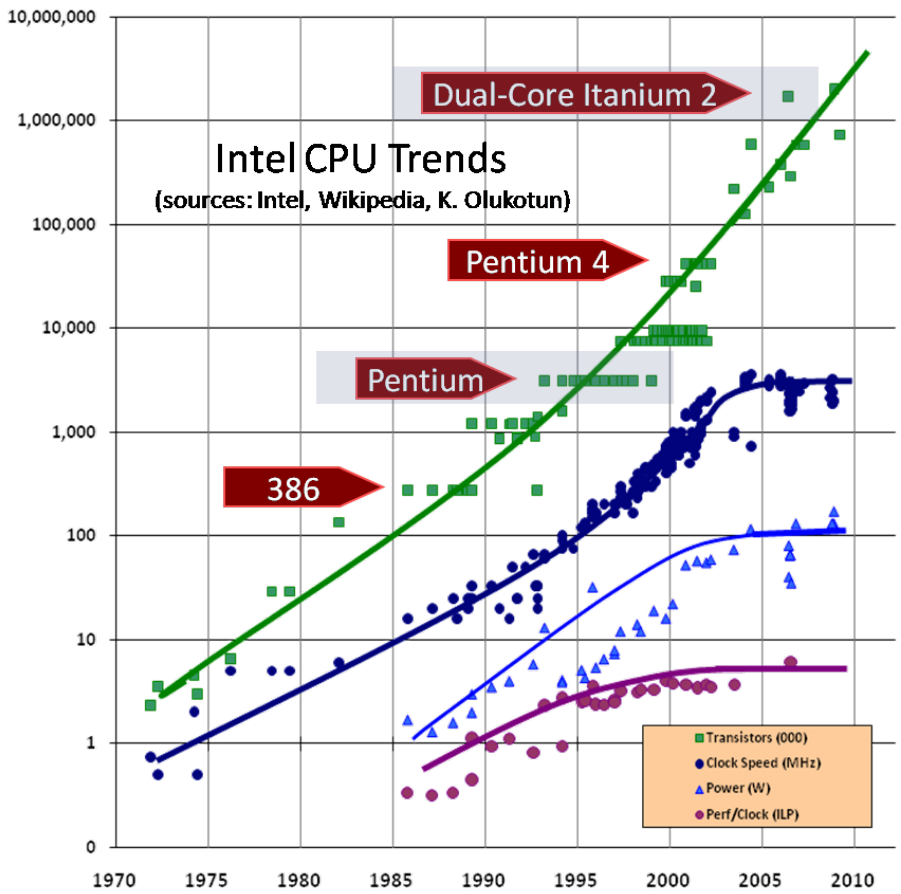


Current Technological Trend



- Implications of Moore's Law have changed since 2003

Current Technological Trend

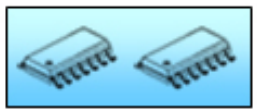


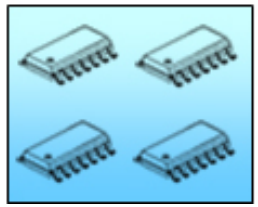
- Implications of Moore's Law have changed since 2003
- 130W is considered an upper bound (the *power wall*)

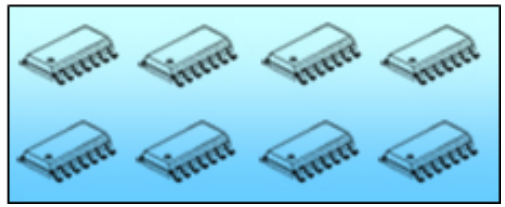
$$P = ACV^2 f$$

Multicore Software Scaling

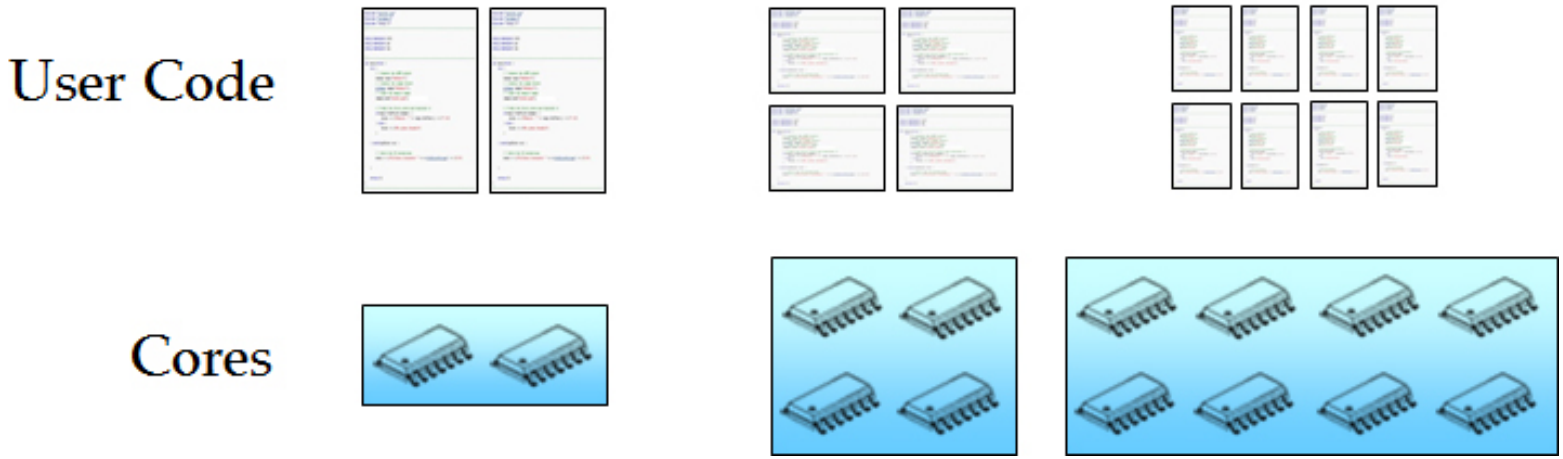
Cores



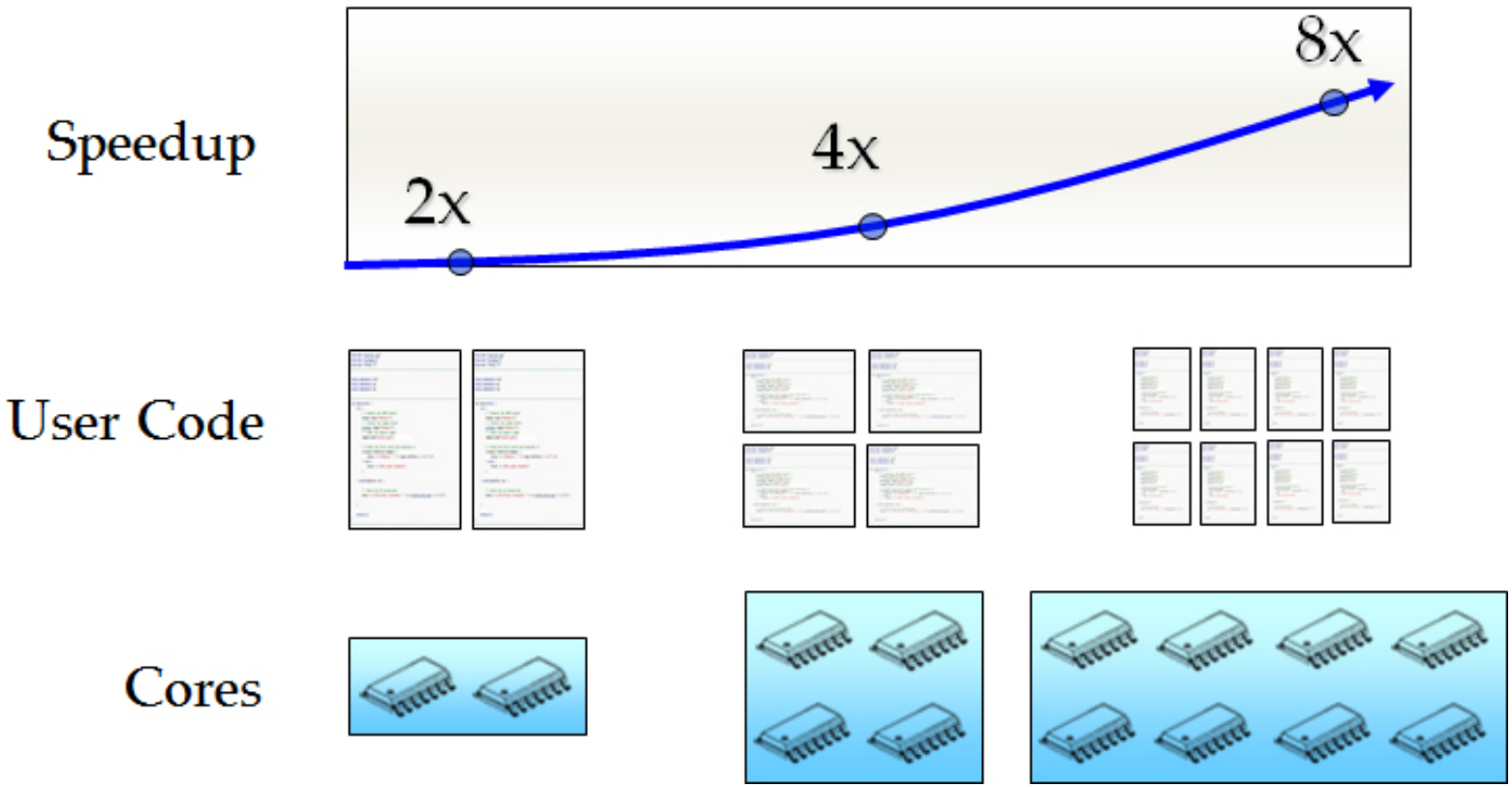




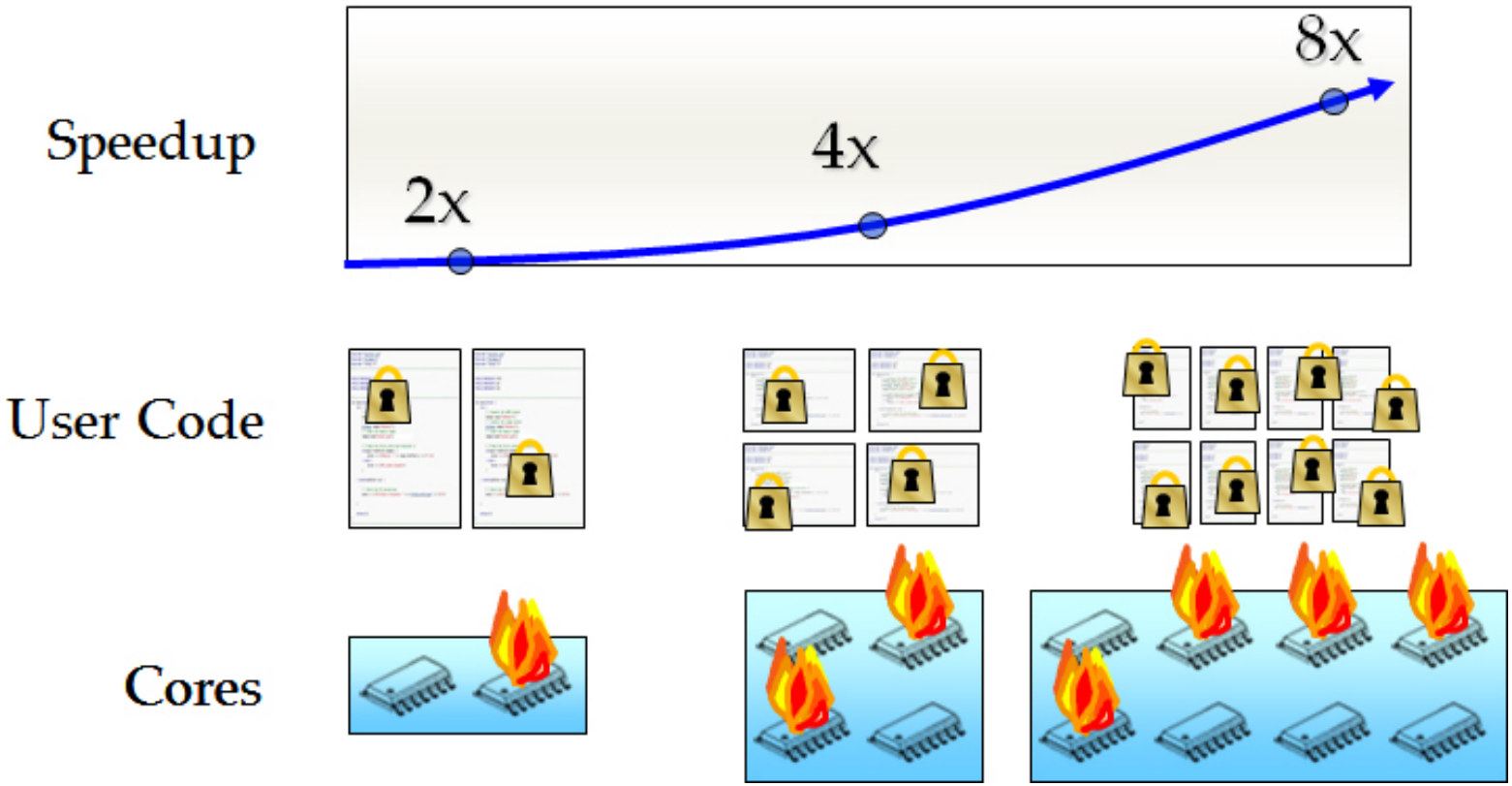
Multicore Software Scaling



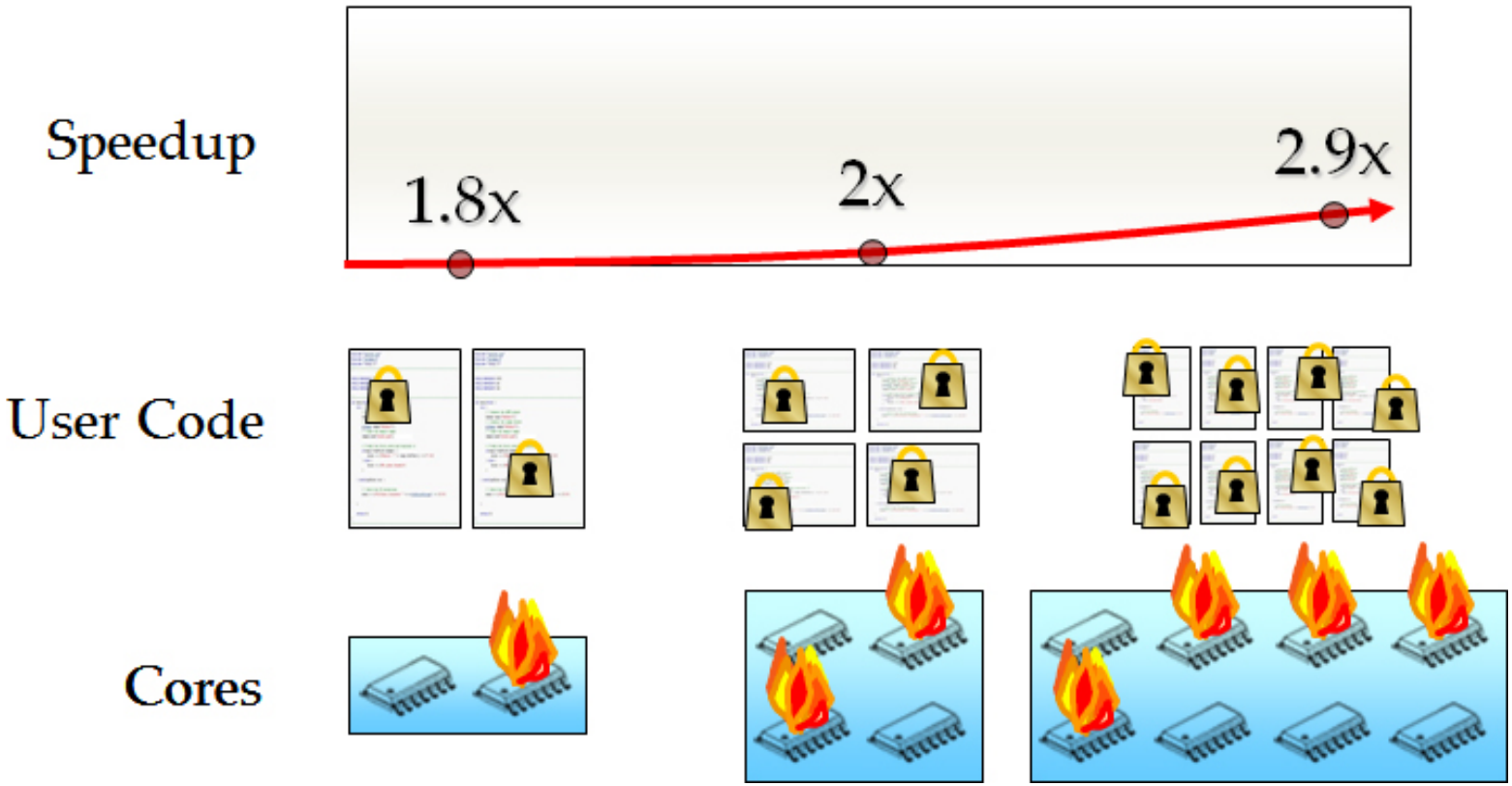
Multicore Software Scaling



Multicore Software Scaling



Multicore Software Scaling



Problem Statement in HPC

- Many HPC applications enforce data separation
  - The amount of synchronization points are already reduced
- Coordination algorithms easily become the bottleneck of HPC Systems on multicore architectures
- They cannot be avoided at all
  - Necessary for the overall correctness
- How to deal with performance issues of HPC Systems on multicores?

Wait-Free Algorithms

- *All threads complete their job in a finite number of steps*

Wait-Free Algorithms

- *All threads complete their job in a finite number of steps*

- Algorithms in which threads communicate using some sort of atomic-update machine instruction
  - Compare and Swap (CAS)
  - Load-Link/Store-Conditional (LL/SC)

- Both CAS & LL/SC will fail if their initial conditions change

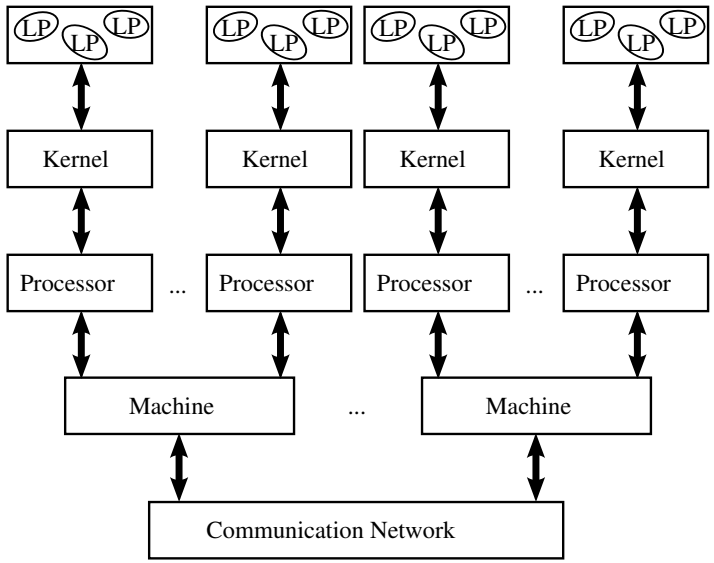
- The "fix" for failing is usually to loop and try again.

Target Architecture: PDES

- Exchanged information unit is the *discrete event*
- The involved objects state is updated according to the flow of events

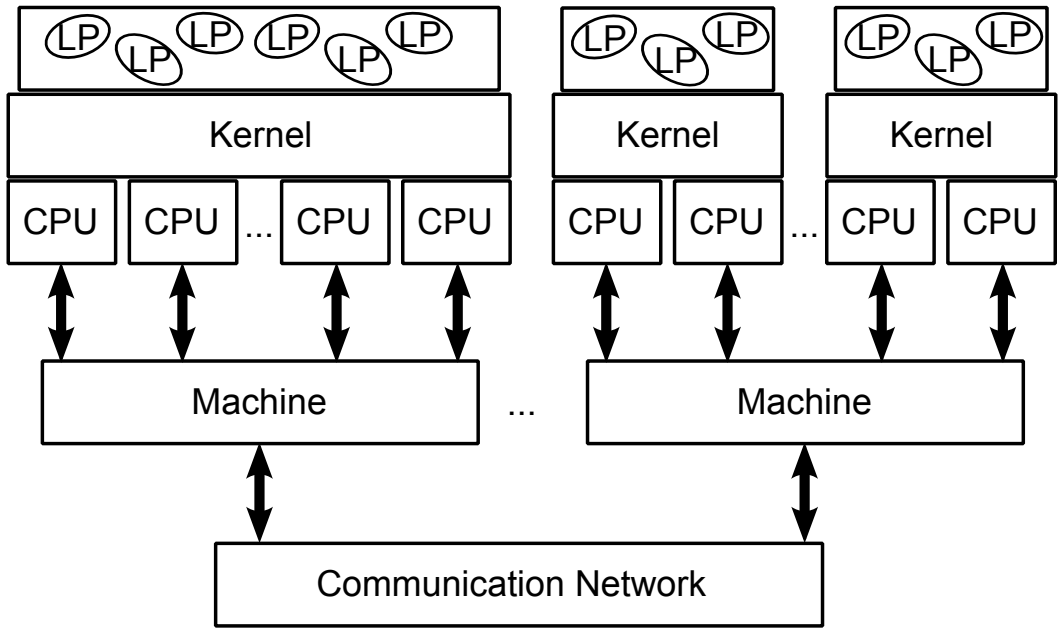
Target Architecture: PDES

- Exchanged information unit is the *discrete event*
- The involved objects state is updated according to the flow of events



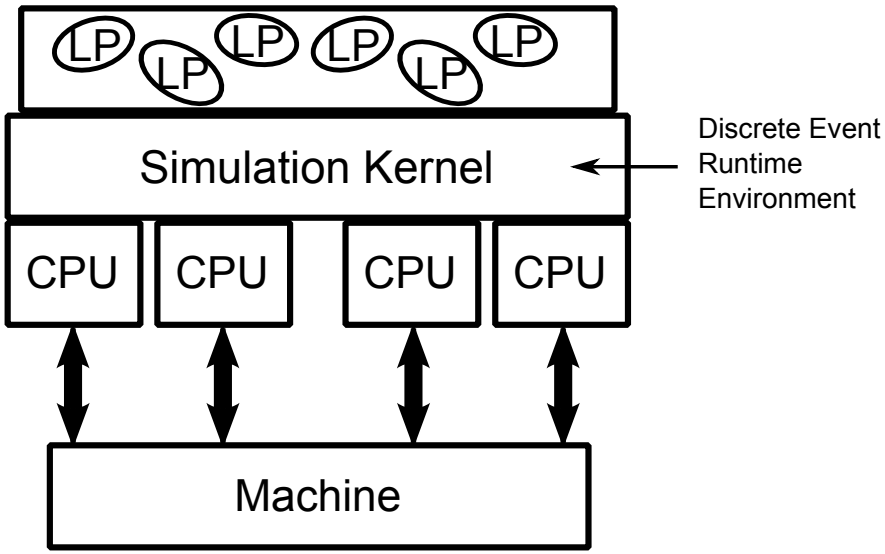
Target Architecture: PDES

- Exchanged information unit is the *discrete event*
- The involved objects state is updated according to the flow of events

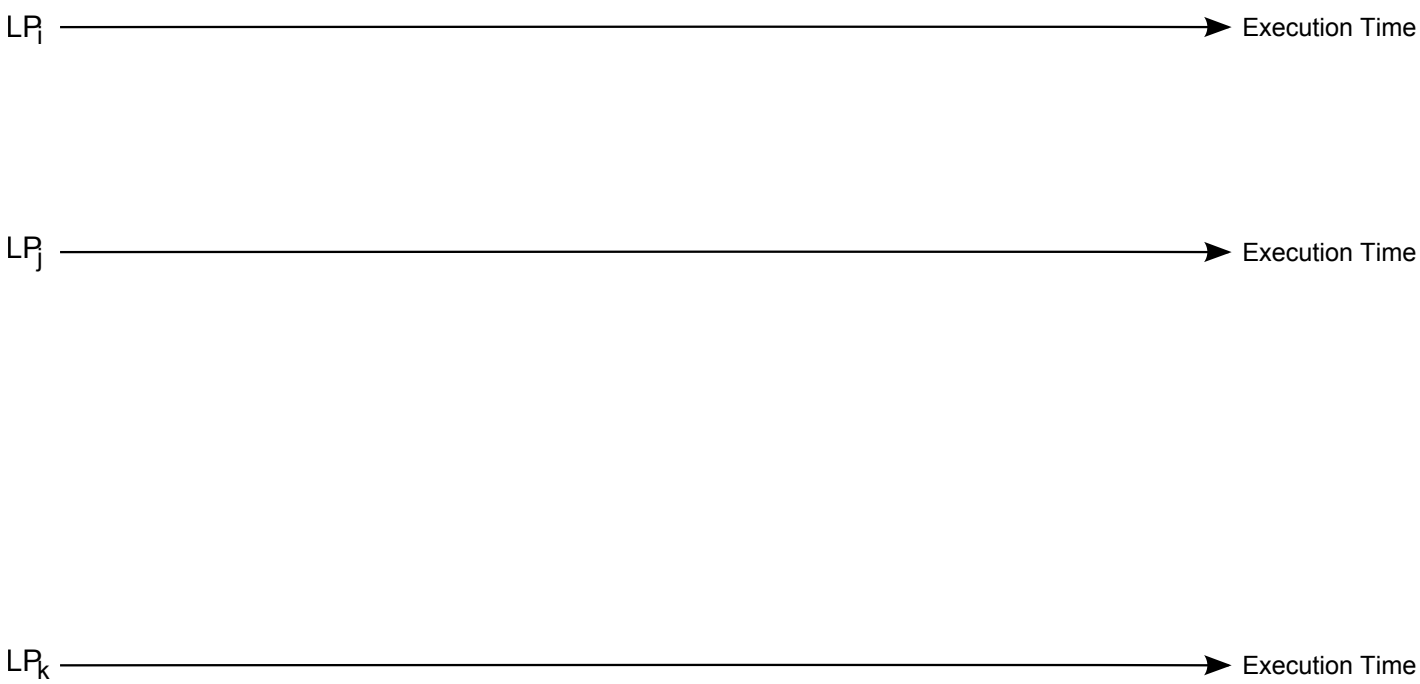


Target Architecture: PDES

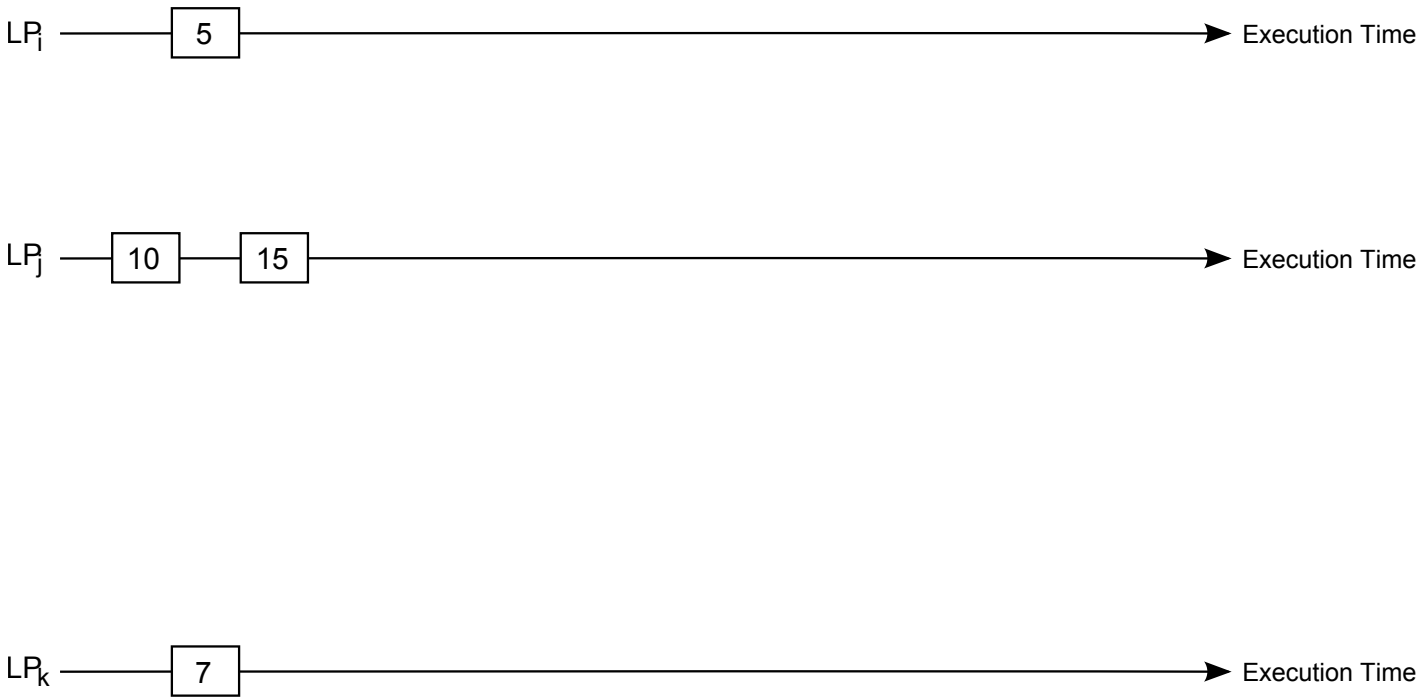
- Exchanged information unit is the *discrete event*
- The involved objects state is updated according to the flow of events



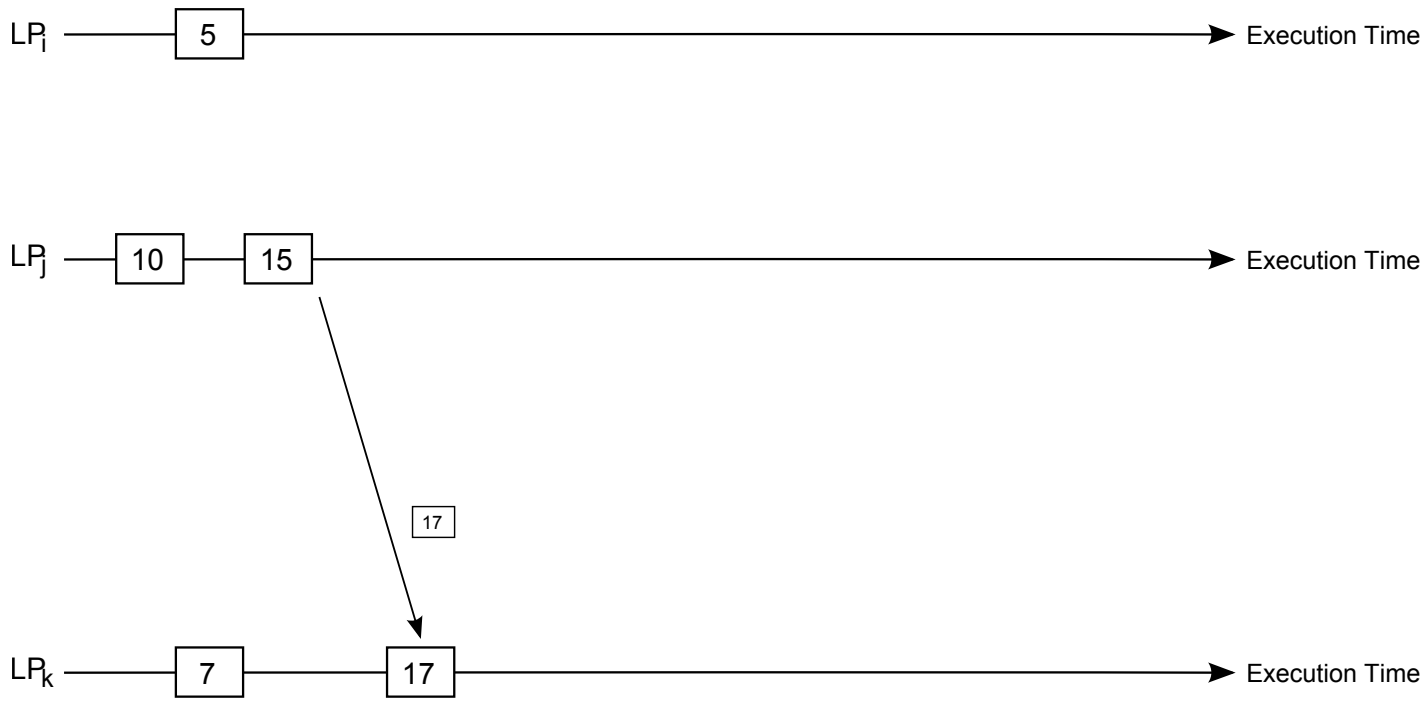
Coordination Algorithm in PDES: GVT



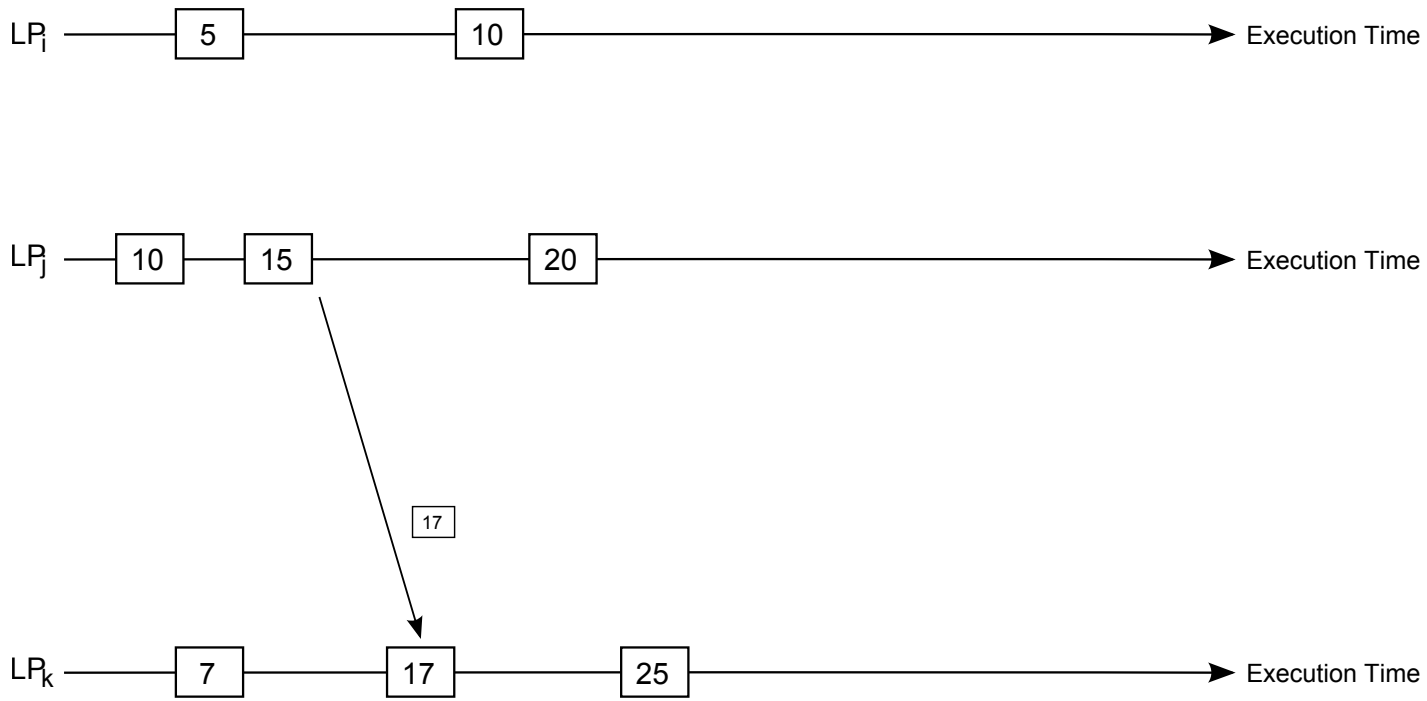
Coordination Algorithm in PDES: GVT



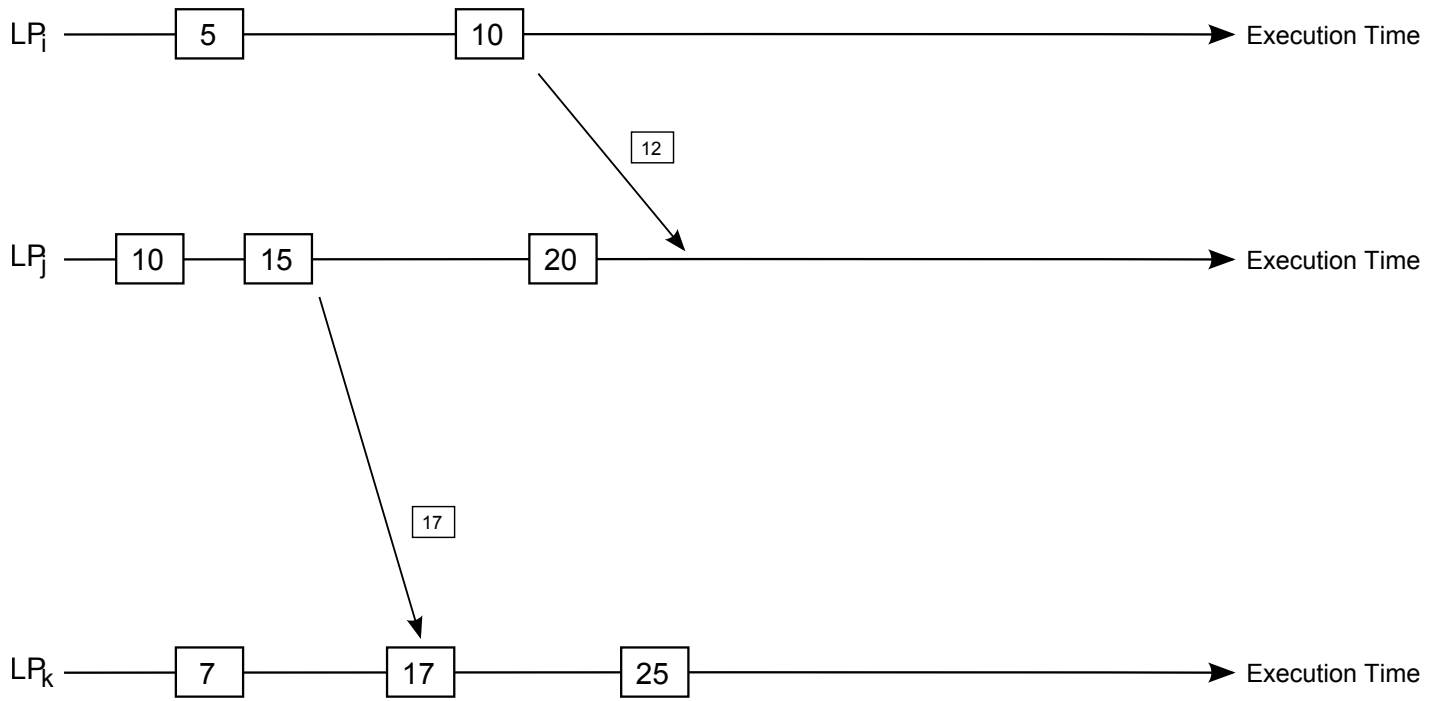
Coordination Algorithm in PDES: GVT



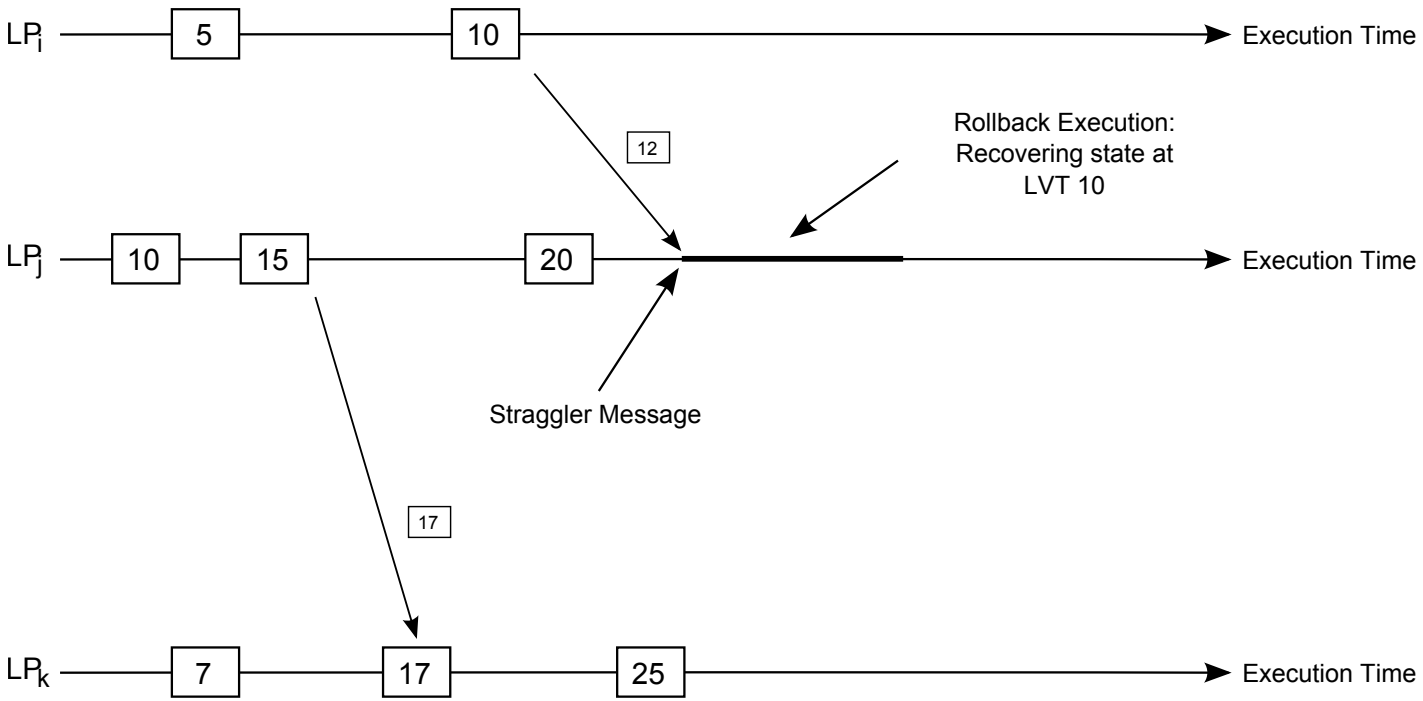
Coordination Algorithm in PDES: GVT



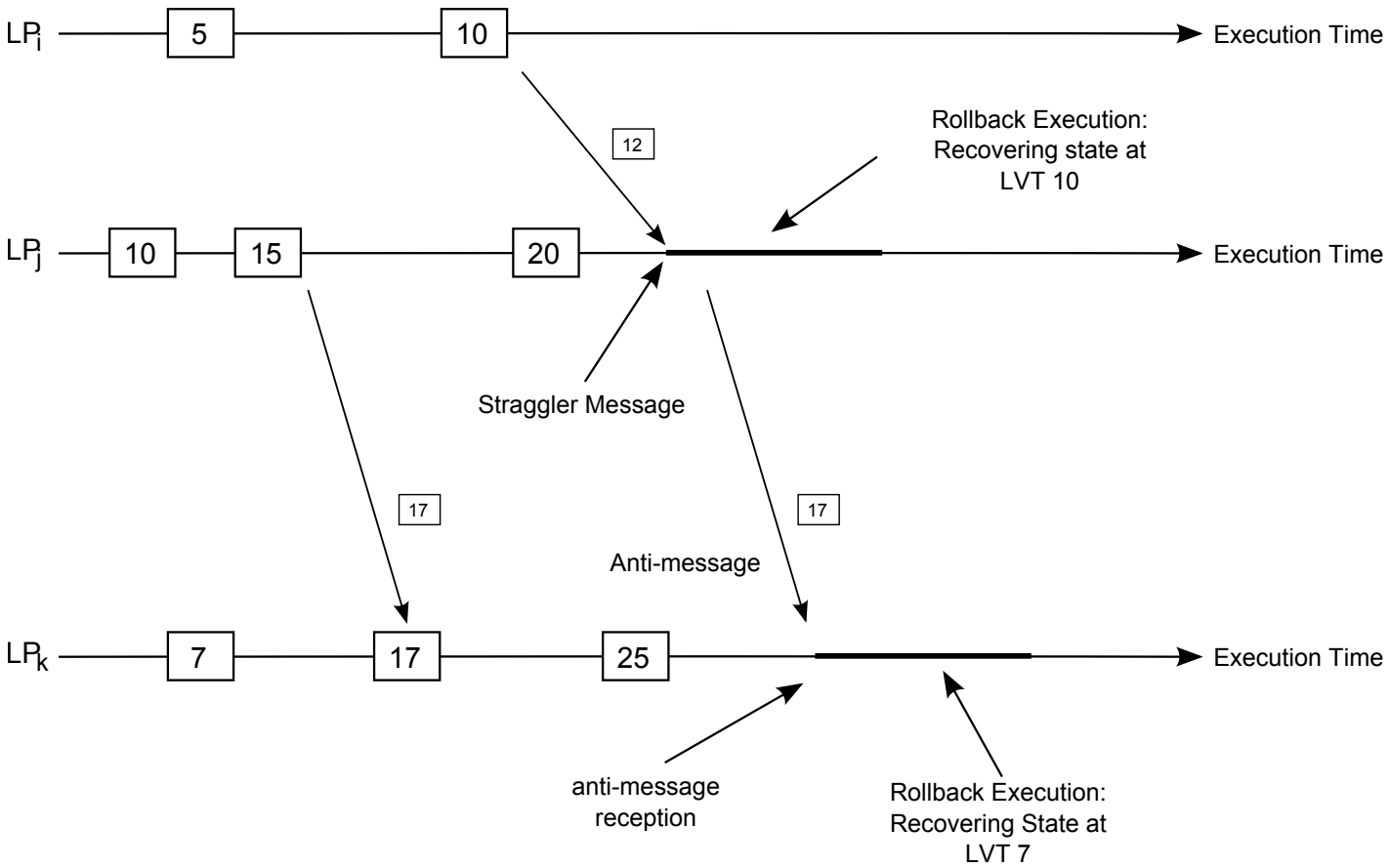
Coordination Algorithm in PDES: GVT



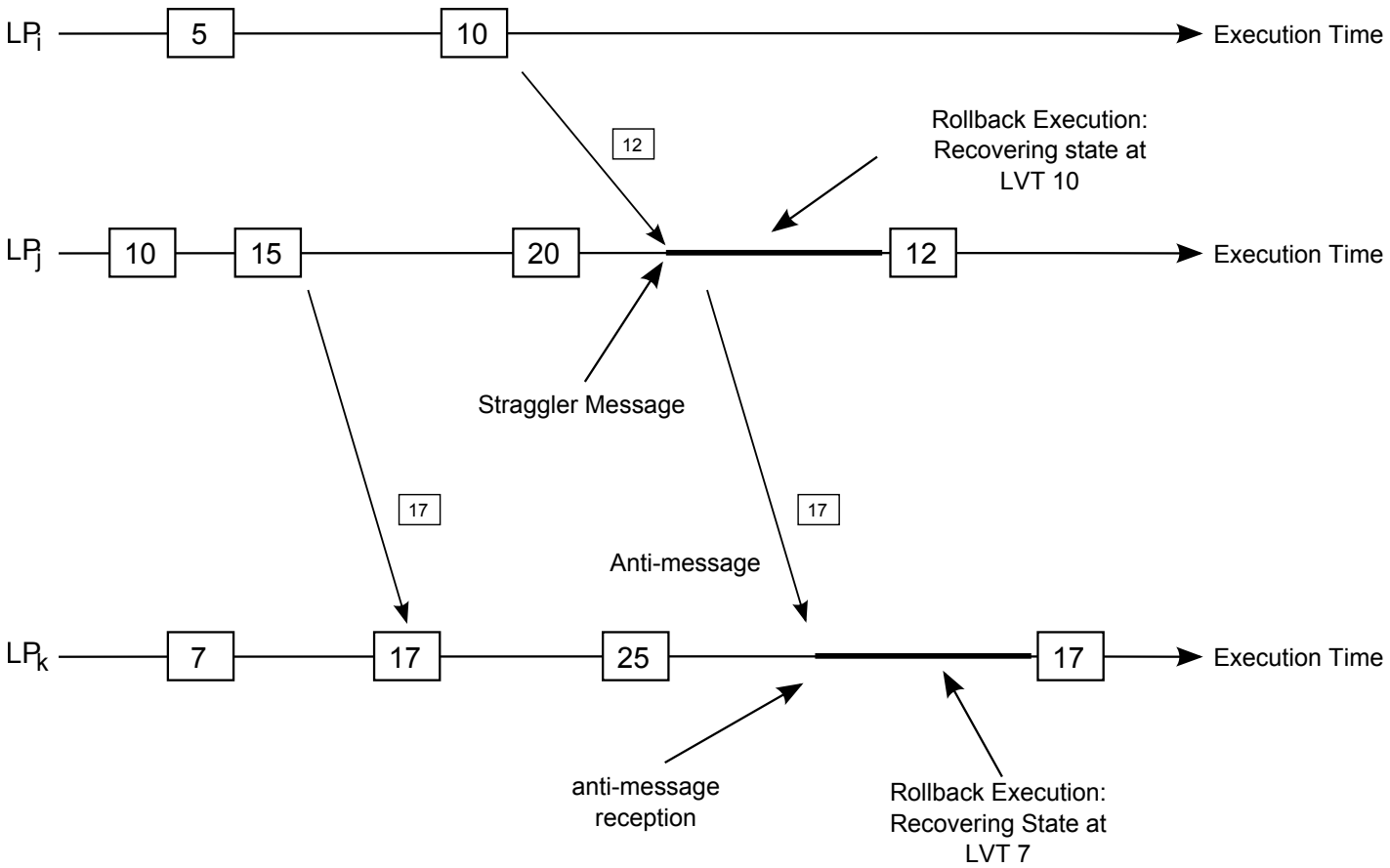
Coordination Algorithm in PDES: GVT



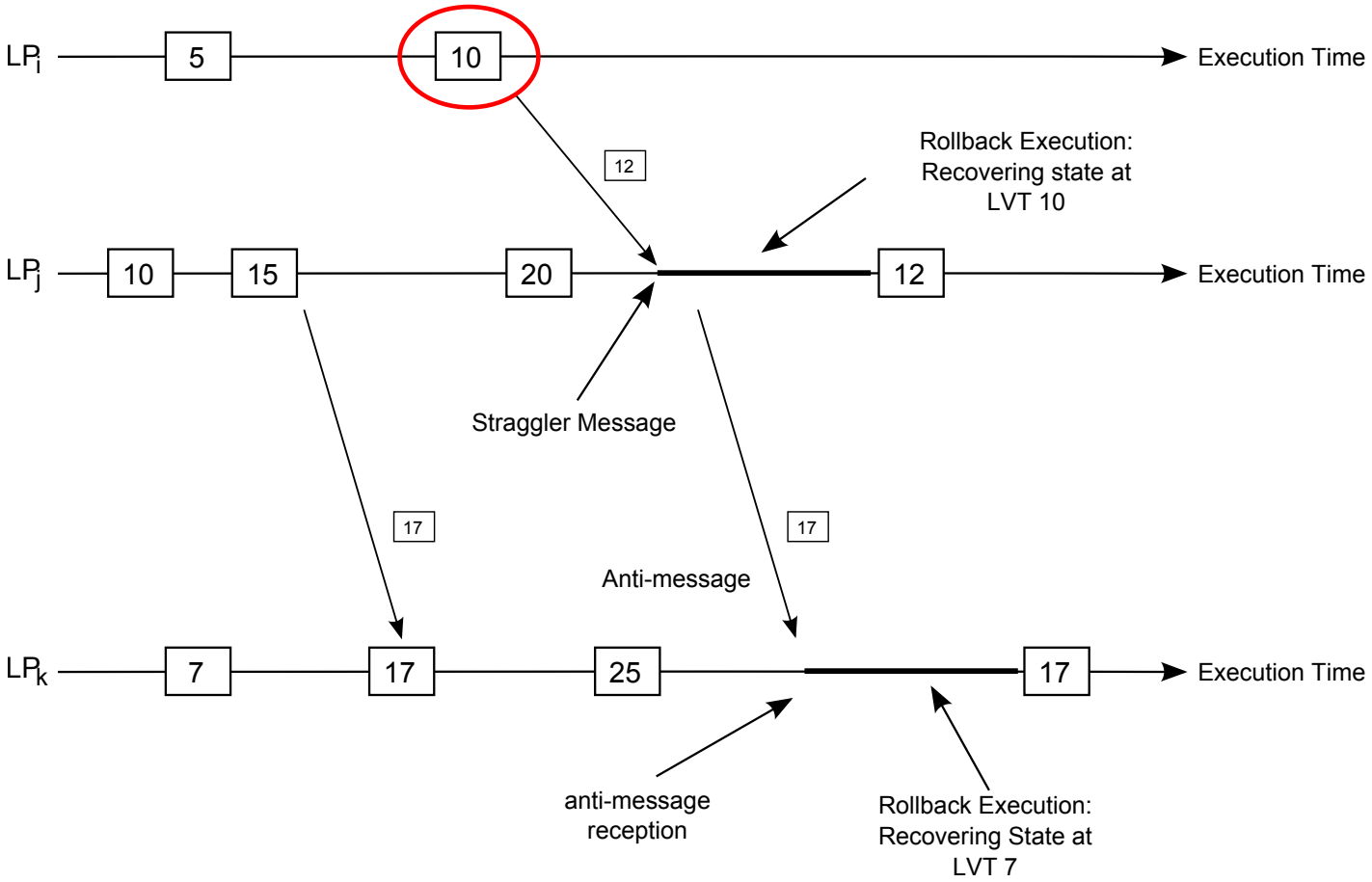
Coordination Algorithm in PDES: GVT



Coordination Algorithm in PDES: GVT



Coordination Algorithm in PDES: GVT



Fujimoto & Hybinette

- Reference algorithm has been so far Fujimoto's and Hybinette's GVT for Shared-Memory Multiprocessors (1998)
- It targets *observable* Time Warp Systems
  - Essentially the send operation leads to the direct incorporation into the message queue
  - This removes the notion of *in-transit* messages
- It relies on the notion of *GVT computation phase*

Fujimoto & Hybinette (2)

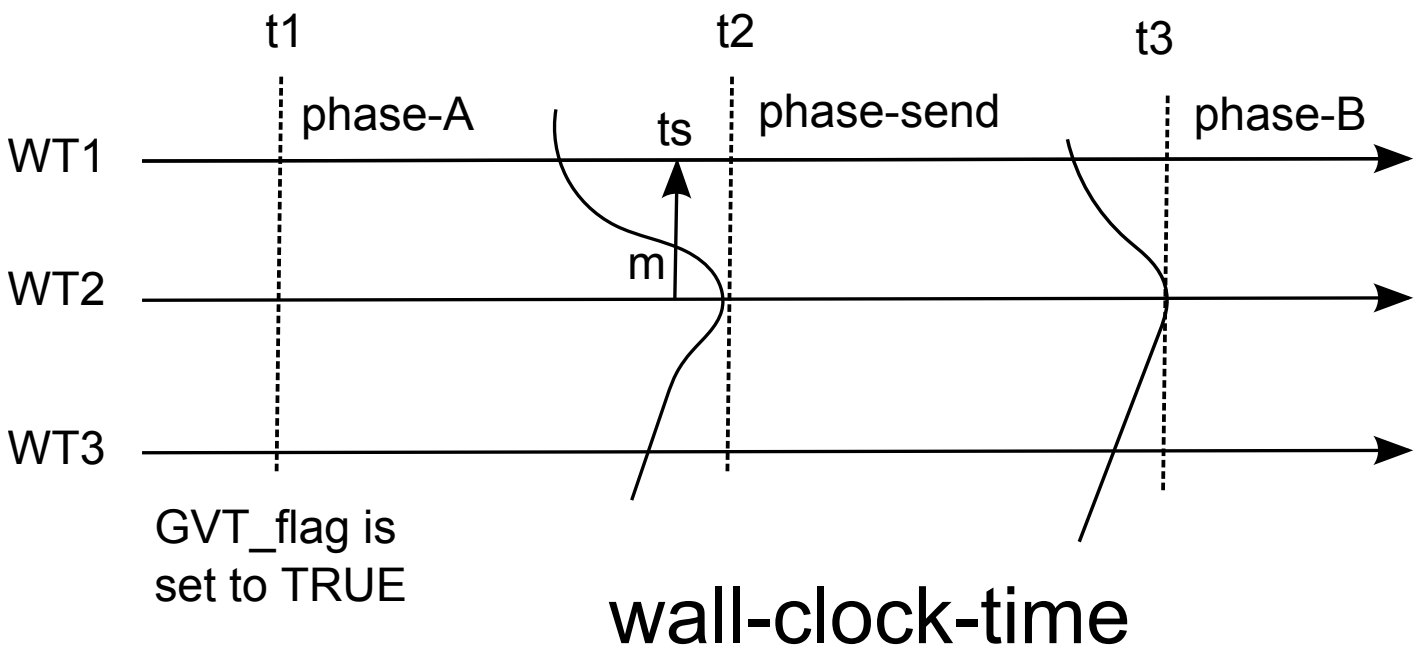
```
procedure MAIN-LOOP
    while there are events to process do
        LocalGVTFlag ← GVTFlag                          ▷ Atomically set to NPE
        Receive messages and process rollbacks
        Receive antimessages, process annihilations and rollbacks
        Select next event M
        if LocalGVTFlag > 0 and LocalMin not computed then          ▷ CS
            PEMin[PE] ← min{SendMin, TS(M)}
            GVTFlags ← GVTFlags - 1
            if GVTFlag = 0 then
                GVT ← min{PEMin[1], ..., PEMin[NPE]}
            end if
        end if
    end while
end procedure
```

Wait-Free GVT Algorithm

- Each worker thread manages a set of LPs
- Event-queues are directly accessible by worker threads
  - No message/anti-message is ever in-flight across worker-threads

- GVT protocol entails determining the right moment to look at data structures and compute the local minimum
  - It is computed after the incorporation of incoming messages/anti-messages into the event-queue
  - No need to account for anti-messages and canceled events timestamp
- The algorithm is wait-free because it relies on a sequence of phases

Wait-Free GVT Algorithm (2)

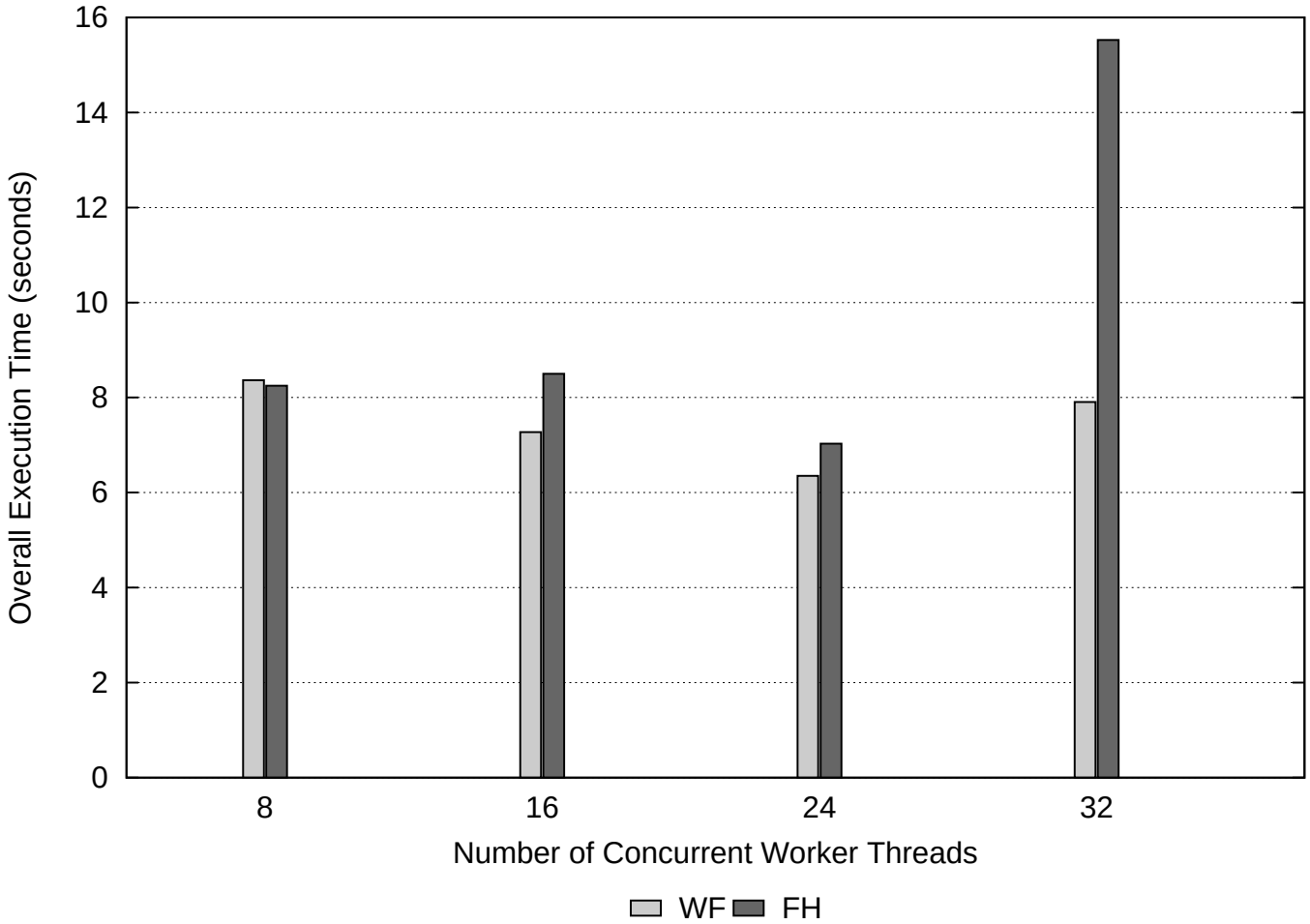


Test-Bed Scenario

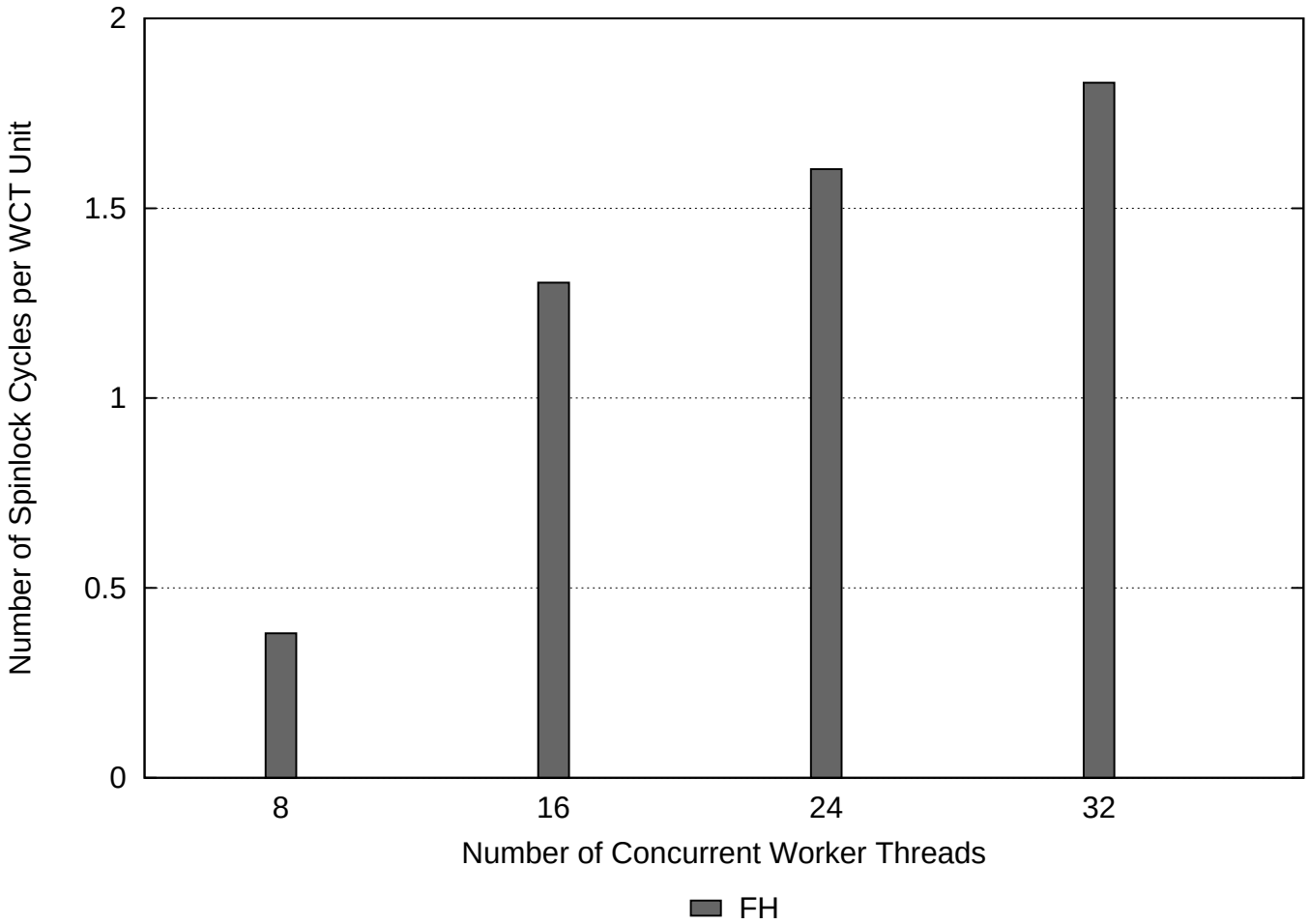
- 32-cores HP ProLiant Server
- 64GB of RAM
- 2.6.32-5-amd64 Linux kernel
- ROOT-Sim Simulation Kernel v. 1.0.2

- Modified version of PHOLD benchmark
  - homogeneous LPs that perform memory allocation/deallocation operations and read/write operations
  - LPs modify their memory layout
  - few and small-granularity events $\Rightarrow$ increased likelihood of competing for the critical section

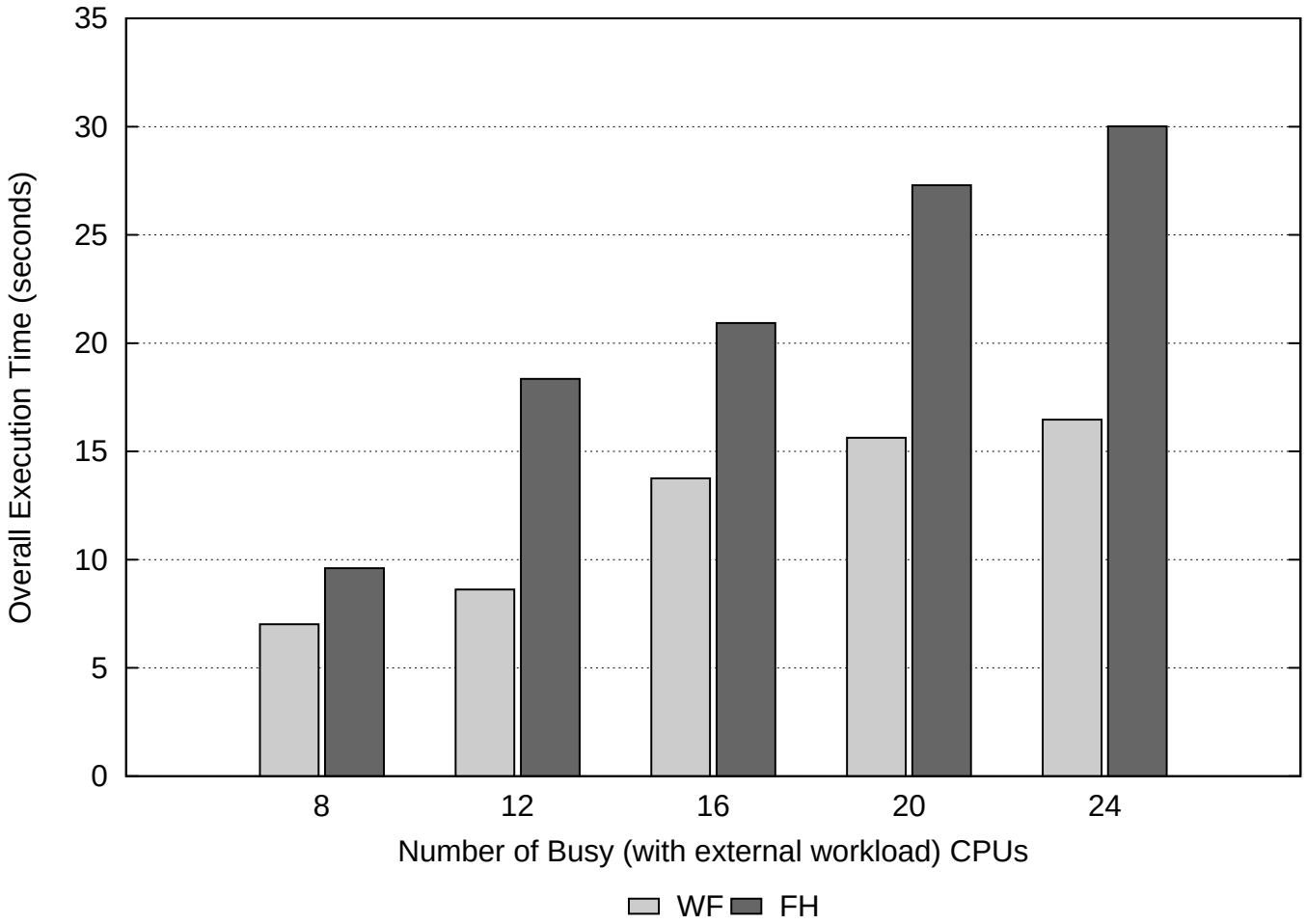
Experimental Results



Experimental Results (2)



Experimental Results (3)



Conclusions

- HPC Applications usually have few synchronization points
  - There is a high demand for enhanced synchronization schemes
- Modern multicore architectures enhance the impact of synchronization points
  - Contention due to locks can hamper scalability
- Wait-free algorithms can be a viable solution to be studied to face this issue
  - They can provide enhanced scalability
  - There is a benefit in non-dedicated environments as well

Thanks for your attention

pellegrini@dis.uniroma1.it
http://www.dis.uniroma1.it/∼pellegrini
http://www.dis.uniroma1.it/∼ROOT-Sim