

# **Managing *and Using* Millions of Threads**

**A New Paradigm for Operating/Runtime Systems**

**Hans P. Zima**

*Jet Propulsion Laboratory  
California Institute of Technology, Pasadena, California*

# Today's High End Computing Architectures do not Effectively Support Parallel Computing

- ◆ *Most current High End Computing architectures are COTS-based.*
- ◆ *Commodity components were developed for a mass market and do not support the specific requirements of parallel computing.*
- ◆ *This has resulted in low efficiency, low-productivity programming models and limitations in the scope of operating systems research.*
- ◆ **Today's standard approaches to HEC architectures, programming models, and operating systems do not appear to be scalable to Petaflops computing and beyond.**

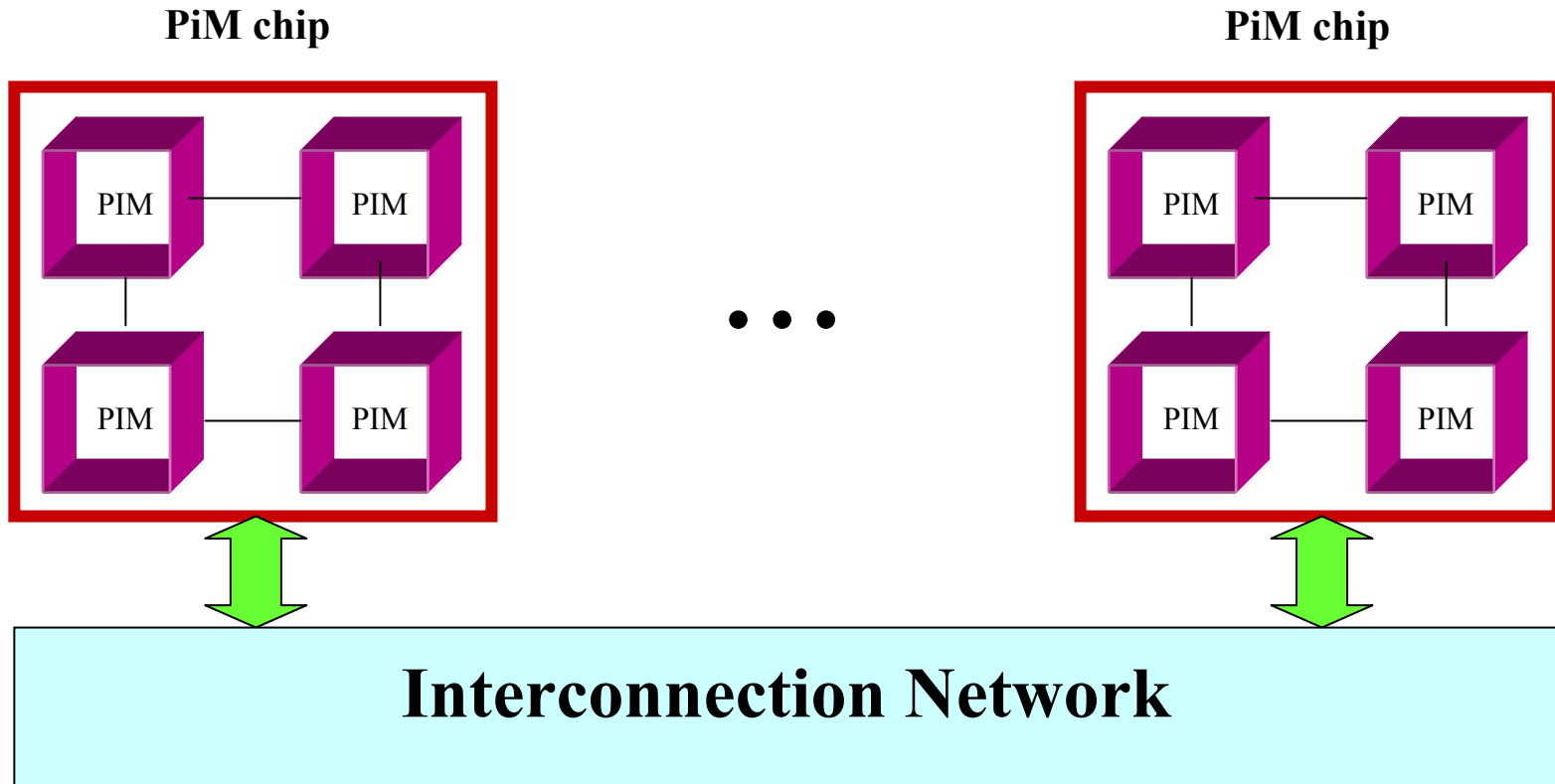
# Architectural Innovation has been Recognized as Critical to HEC

**Architectures emerging around 2010 will deliver Petaflops performance via massive parallelism -- providing millions or even billions of threads.**

**This will not just be an evolutionary development but lead to a revolutionary change of paradigm.**

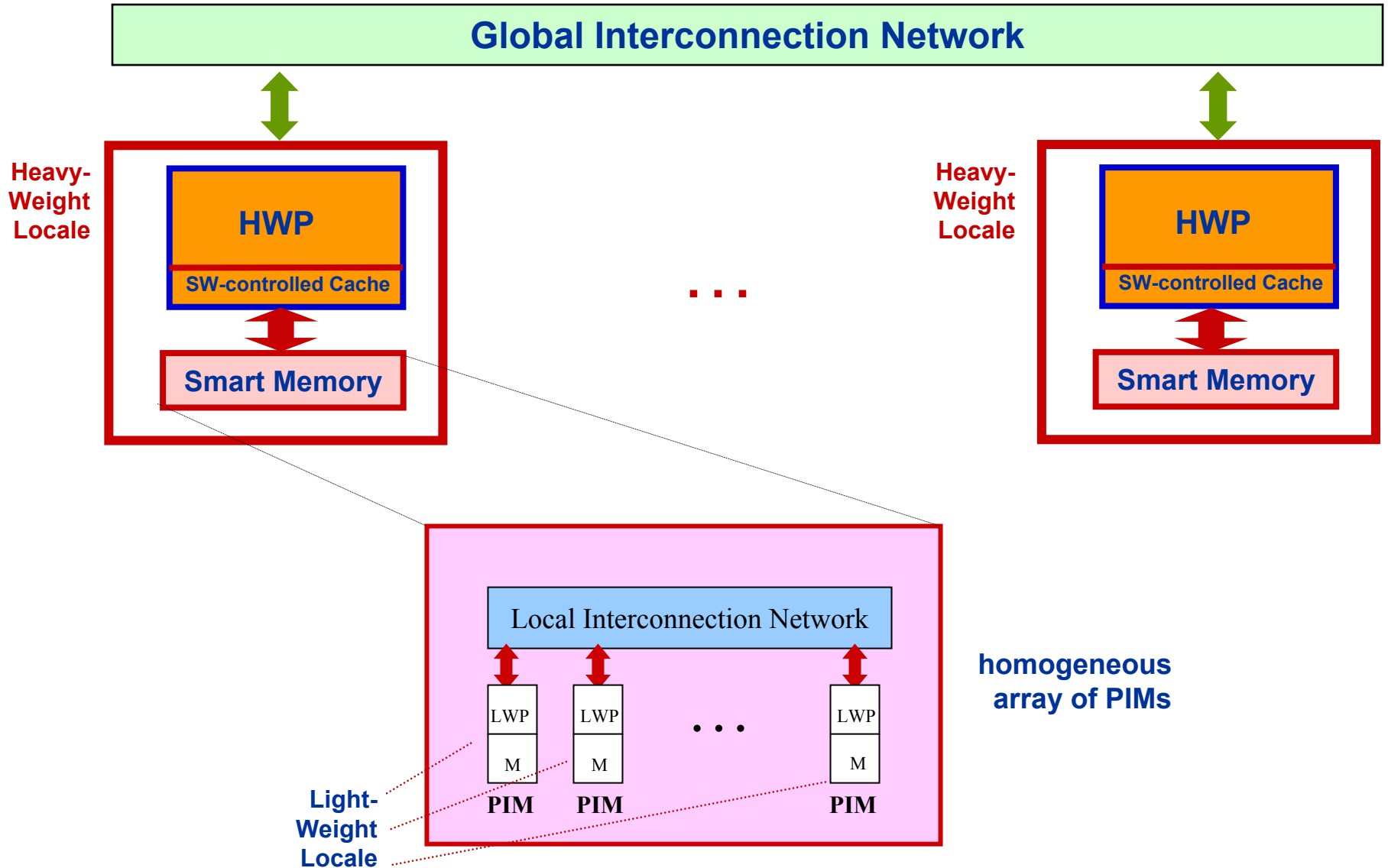
**Two architecture examples...**

# Homogeneous PIM Arrays



**Examples:** *IBM Blue Gene BG/C, PiM-Lite, Gilgamesh*

# Cascade Global Hardware Architecture



# Some Aspects of Massive Parallelism

- ◆ There may be more potential parallelism in the system than can be exploited at a given time.
- ◆ Relative cost of (light-weight) processors is shrinking – their full exploitation may no longer remain top system priority.
- ◆ Management of parallelism must be dynamic, adapting to actual requirements.
- ◆ Structure of parallelism will generally be hierarchical, combining homogeneous and heterogeneous substructures.
- ◆ **Key Issues: the necessity to deal with**
  - *hardware and software faults*
  - *complexity*
  - *locality*

# Massive Parallelism: Major Research Directions

- ◆ **Parallel algorithms and applications**
- ◆ **Execution models, programming paradigms and languages**
- ◆ **OS/Runtime issues: how to support application execution**

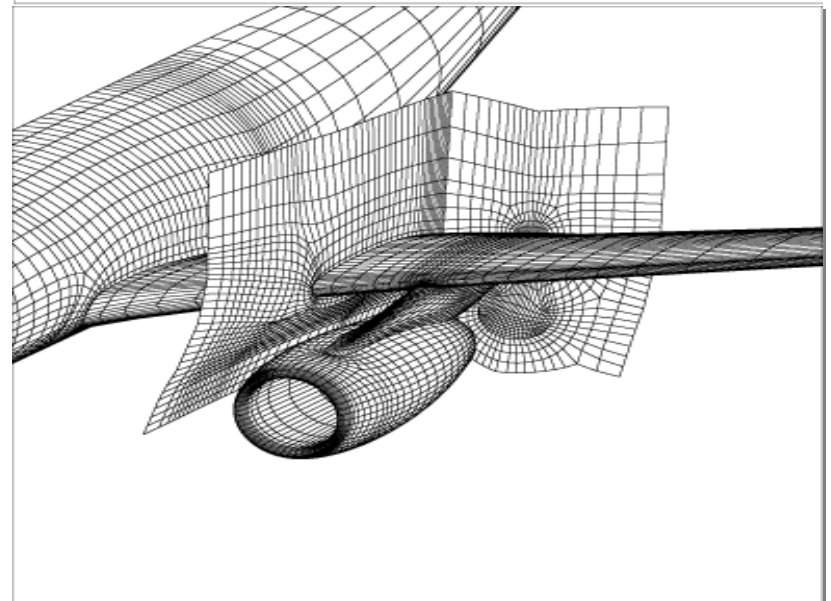
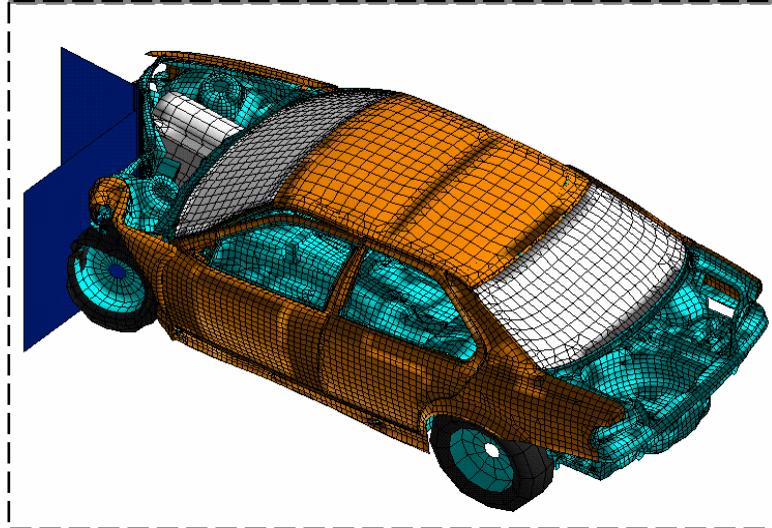
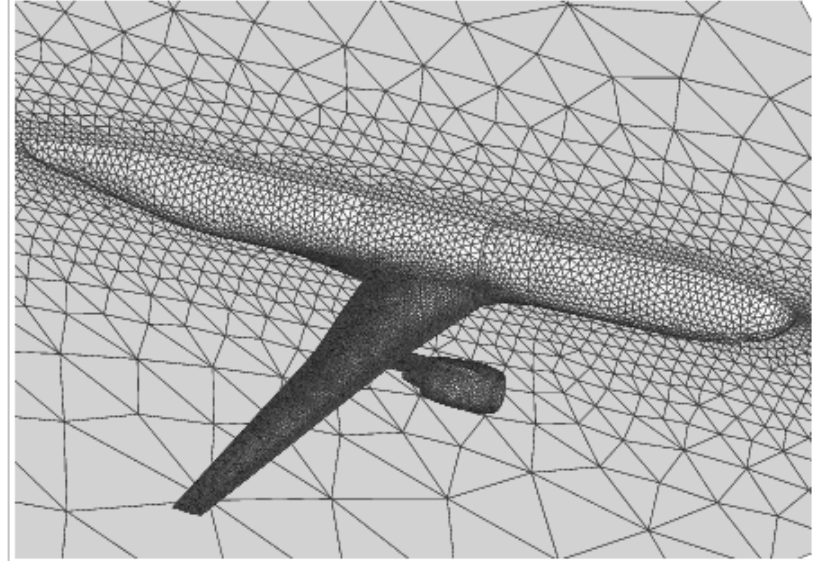
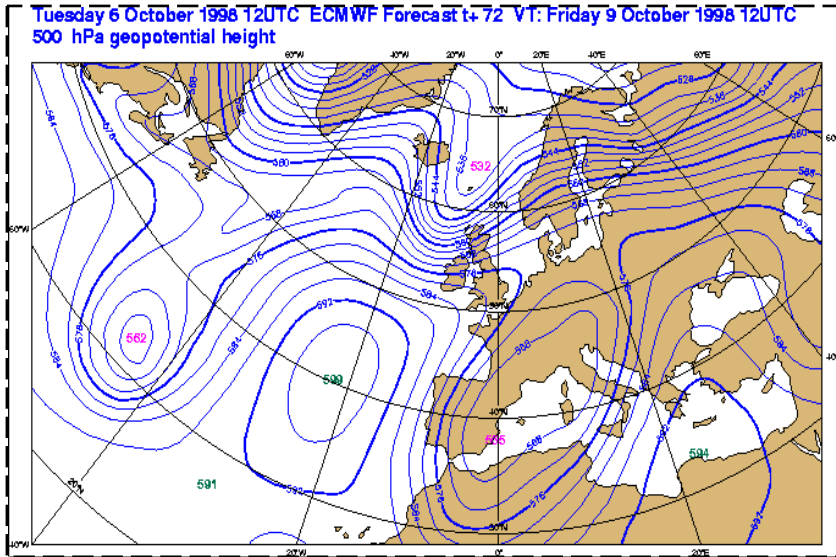
# Massive Parallelism for Algorithms: PRAM Examples

- ◆ **Dense matrix vector multiply:**  $O(\log n)$
- ◆ **Compacting a sparse array (prefix sums):**  $O(\log n)$
- ◆ **Tree compaction (expression evaluation):**  $O(\log n)$
- ◆ **Maximum search:**  $<O(\log \log n)$

Can these complexities be reached?

2  
...  
1  
T  
1

# A Range of Other Applications ...



# Massive Parallelism and the OS/Runtime System

## ◆ OS/runtime system must support

### – scalability

### – fault prevention and fault tolerance

- ◆ *fault anticipation*

- ◆ *graceful degradation*

- ◆ *effective hierarchical recovery mechanisms (analogy to biological systems!)*

### – **autonomic introspective hardware/software infrastructure for security and self-optimization**

- ◆ *intrusion detection, prevention, and recovery*

- ◆ *load balancing and locality management*

- ◆ *performance analysis, prediction, and feedback-oriented tuning*

# Hardware-Supported Thread Systems

## ◆ Light-weight threads

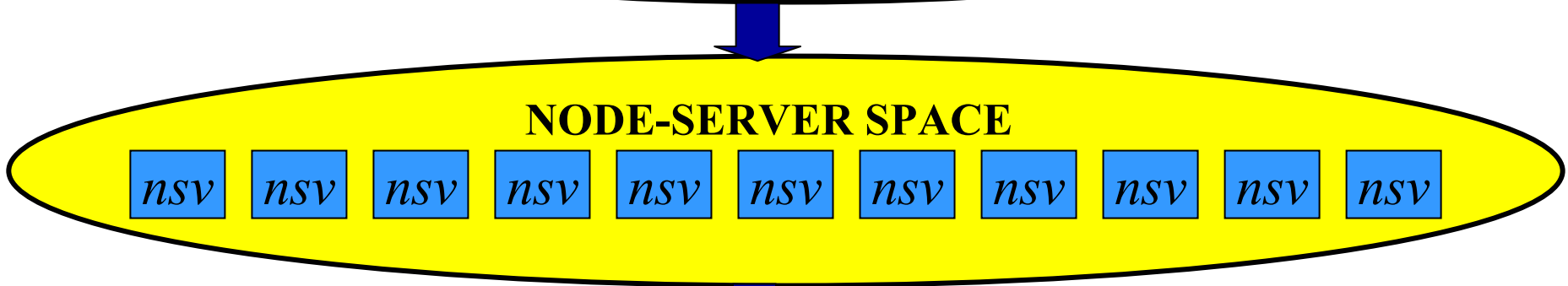
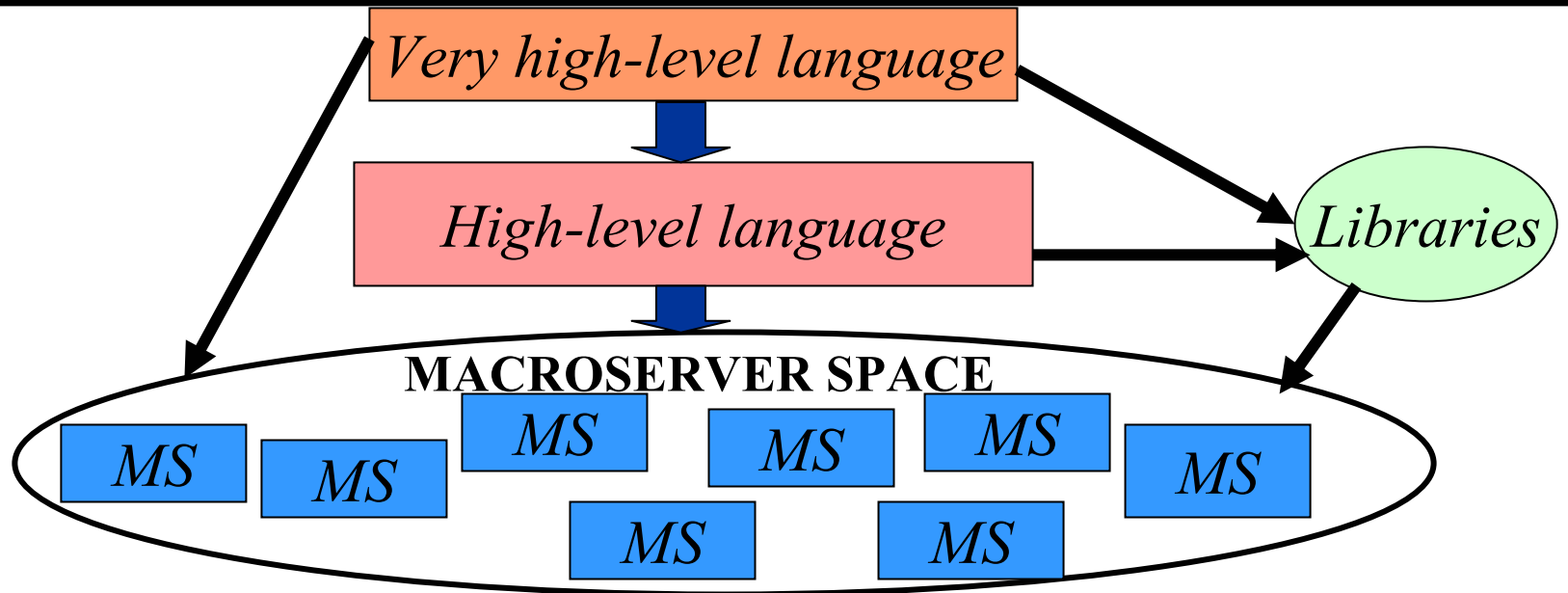
- *small state*
- *mobile and **locality-aware***

## ◆ Thread groups

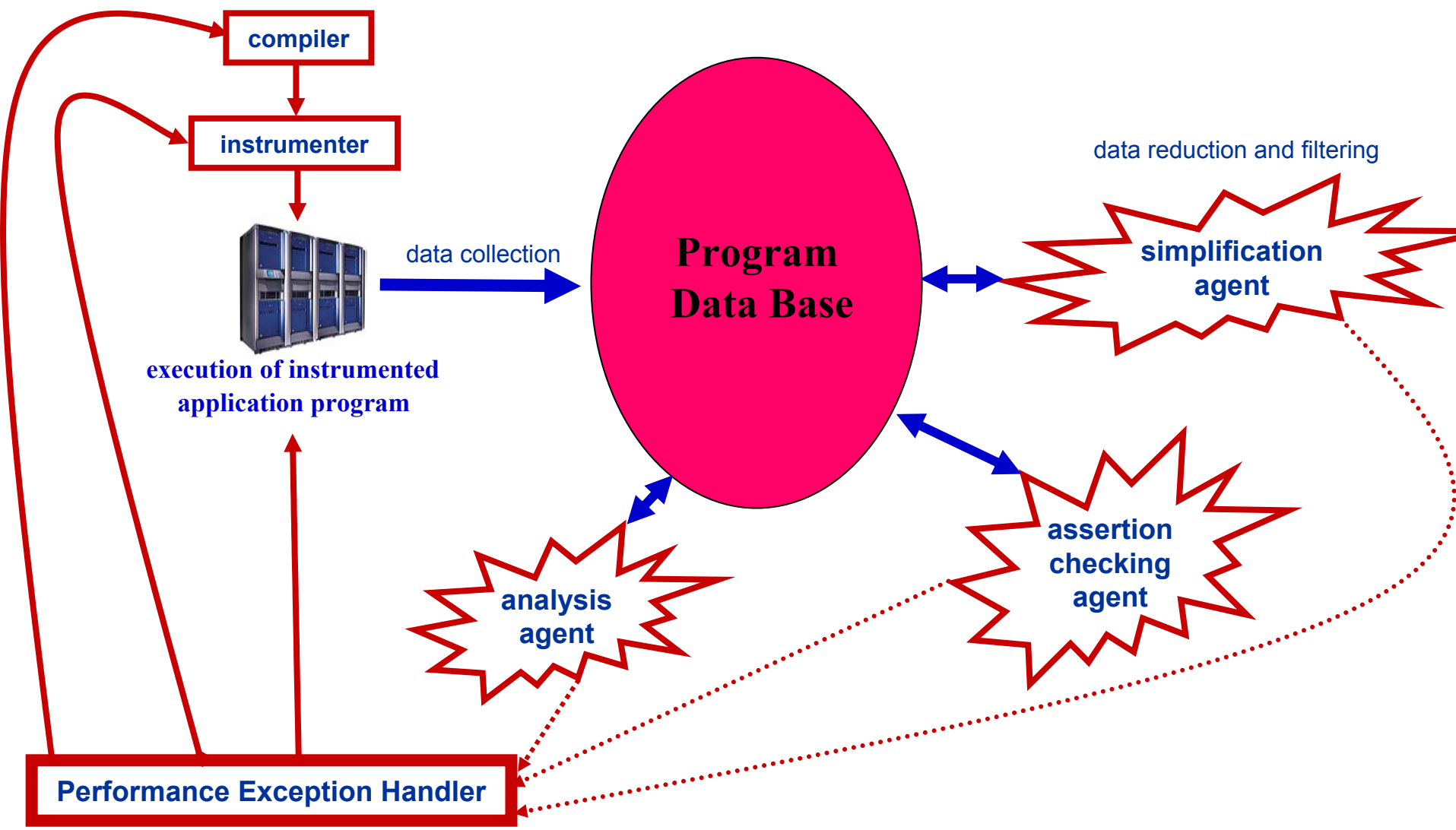
- *dynamically managed*
- *data parallelism support*
- **key: efficient collective communication for regular and irregular problems**
  - ◆ *initiation and termination of parallel activities*
  - ◆ *multicast, prefix operations, reductions, all-to-all communication ... (cf. MPI)*
  - ◆ *redistribution and realignment*
- *association with data structures (**affinity**): binding threads to individual data elements or substructures*

## ◆ Hierarchical structuring of thread systems (macroserver)

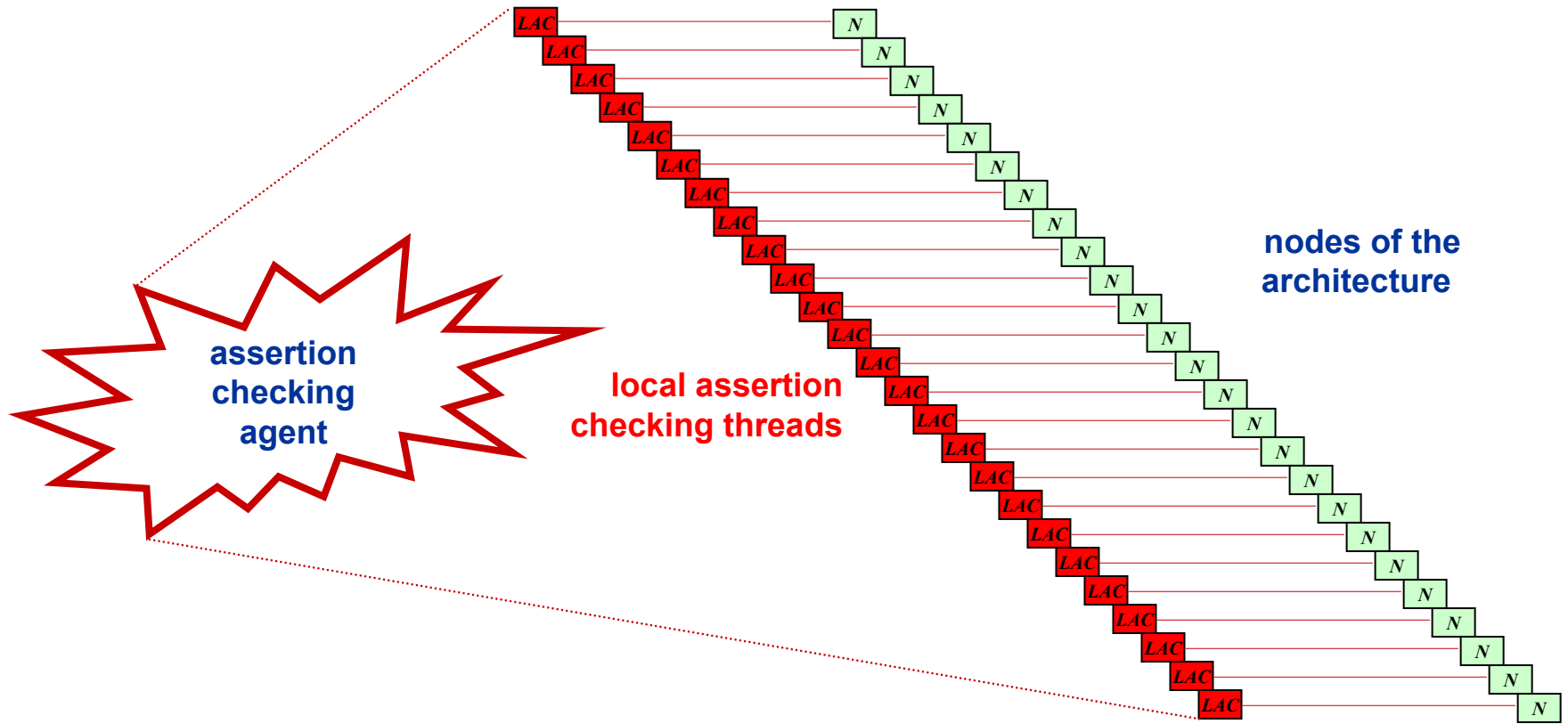
# The Gilgamesh Software Structure



# Case Study: A Society of Agents for Performance Analysis and Feedback-Oriented Tuning



# Agents Can be Massively Parallel



# Conclusion

- ◆ **Conventional approaches to HEC architectures are not perceived as sufficiently scalable to meet future requirements**
- ◆ **Petaflops systems emerging at the end of this decade are expected to efficiently support massively parallel thread systems**
- ◆ **OS/runtime systems must deal transparently with**
  - *fault management*
  - *security*
  - *complexity and performance: autonomic self-optimization*
- ◆ **Effective OS/runtime solutions to be seen in the context of research in algorithms, execution models, programming paradigms, and languages for massive parallelism**