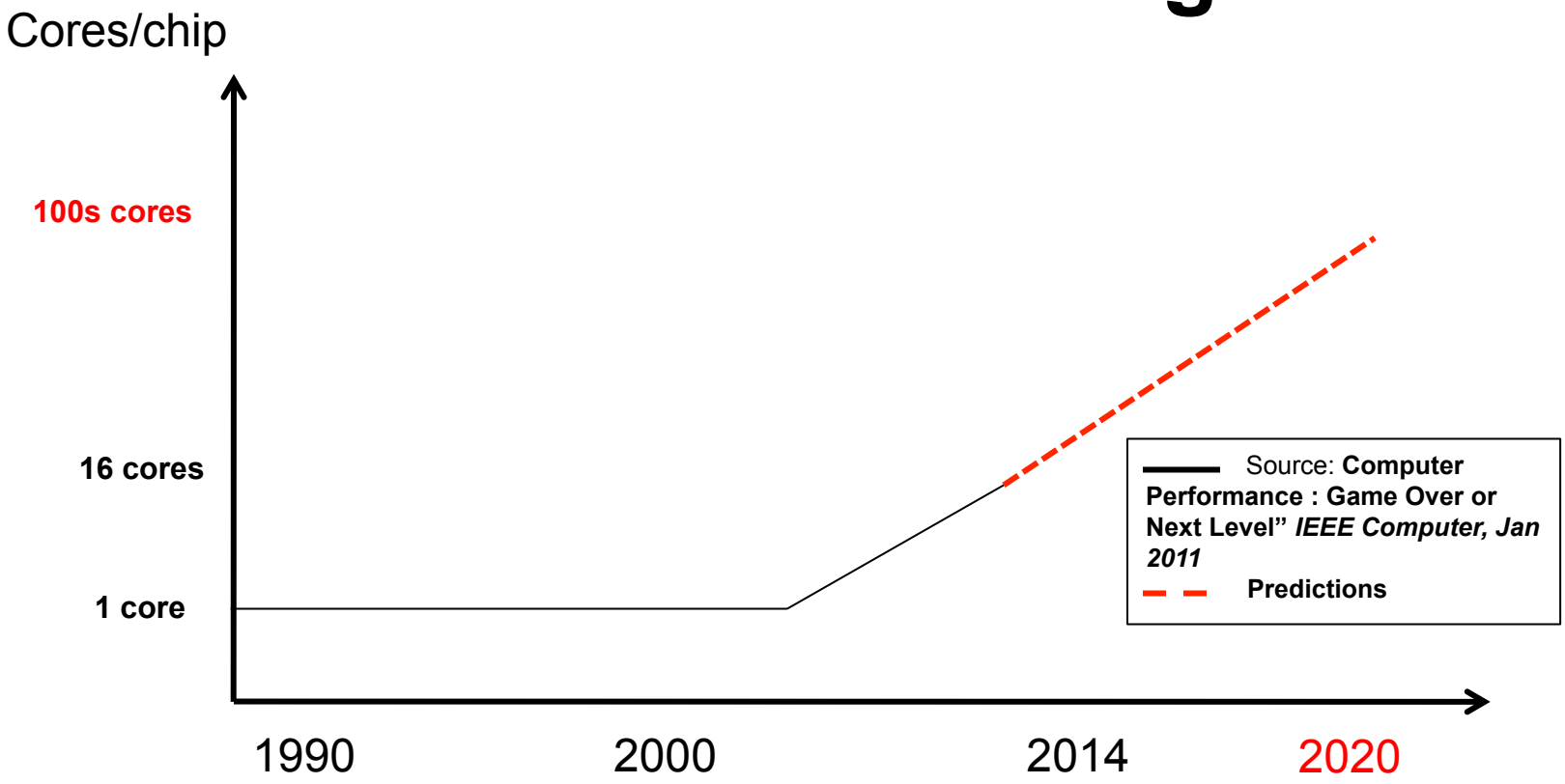
Programmability in the Era of Parallel Computing

Per Stenström

Department of Computer Science and Engineering
Chalmers University of Technology
Sweden

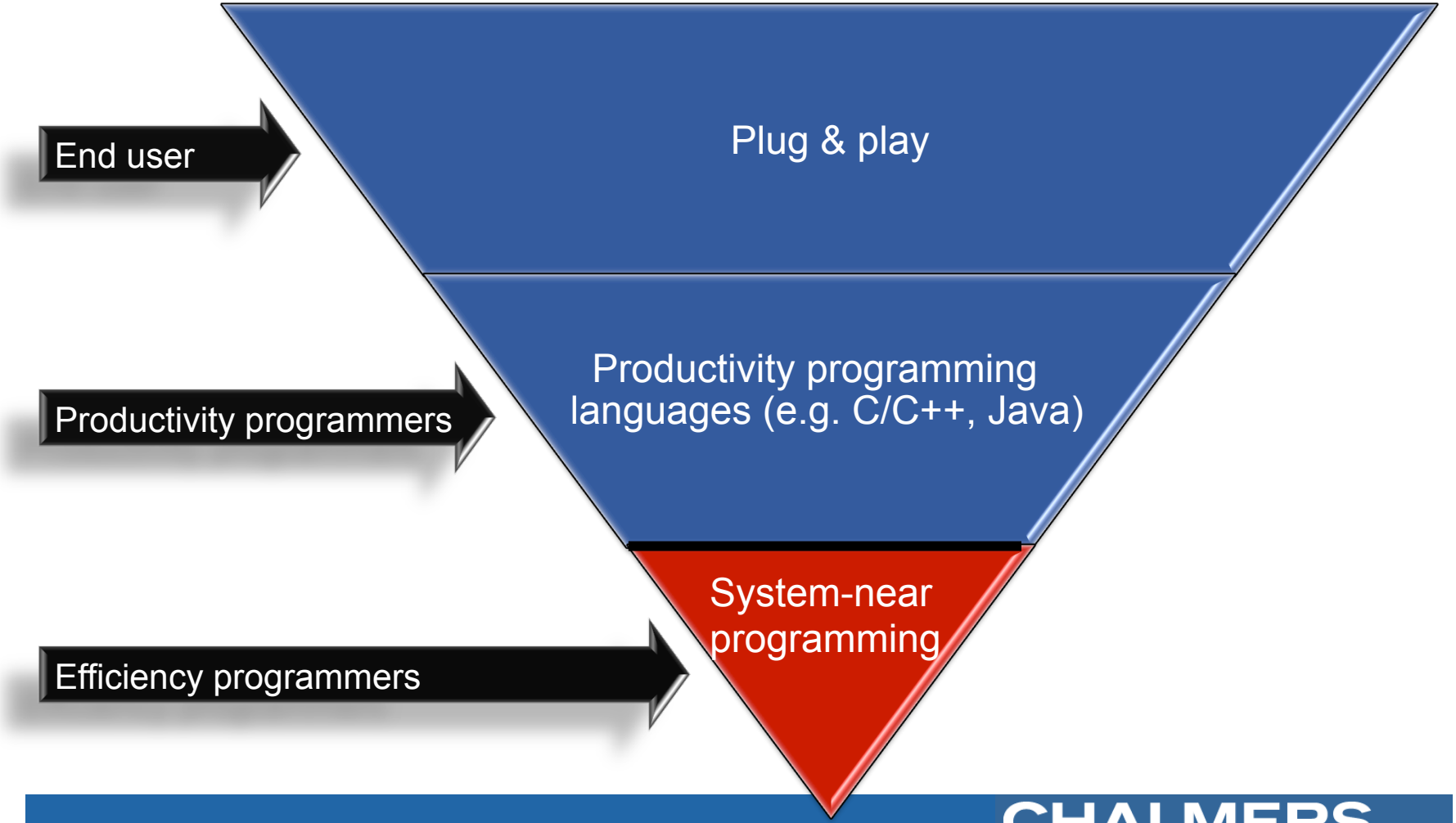
Multicore Scaling



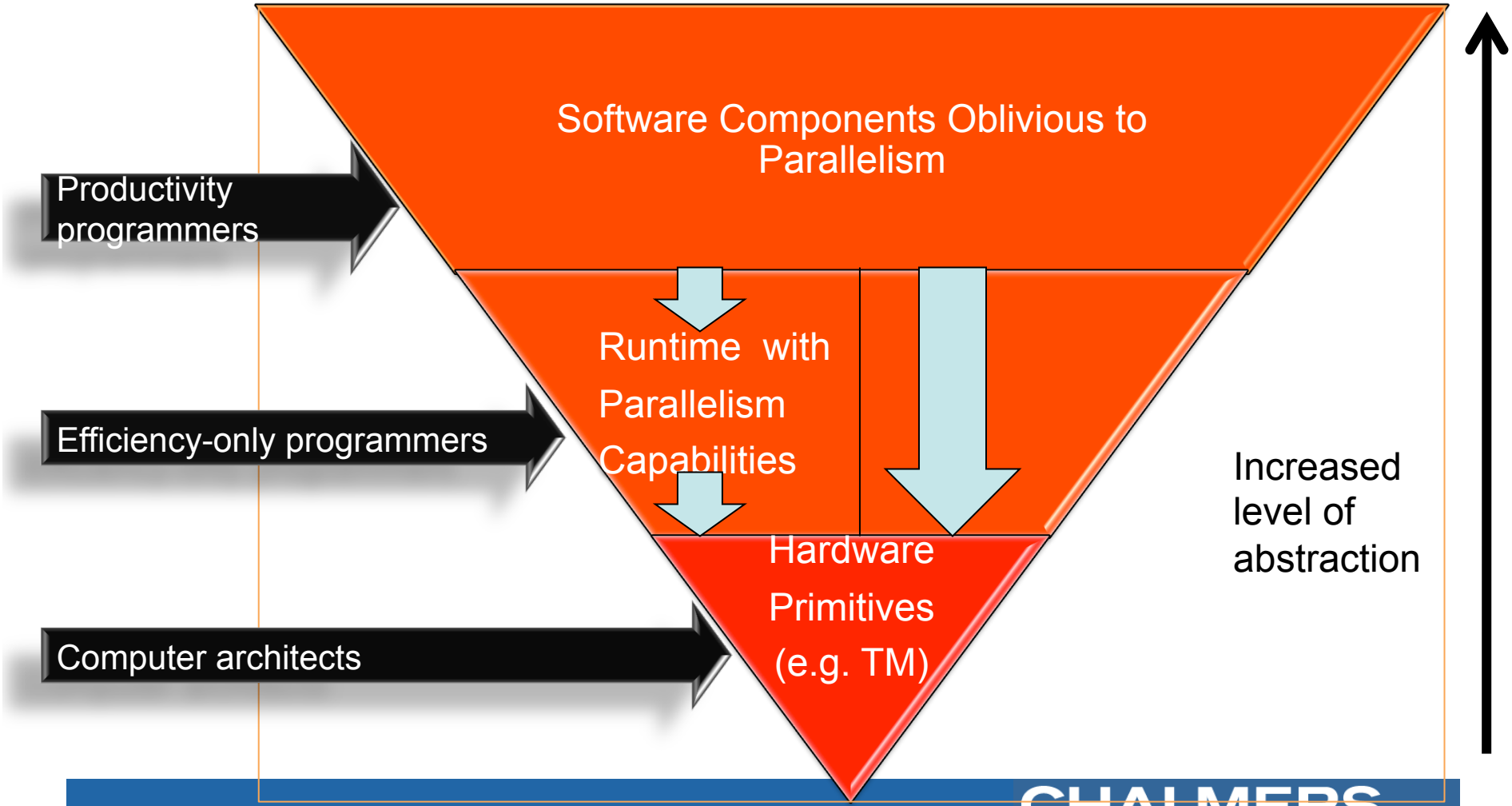
By 2020, several hundreds of powerful cores/chip

Programmability

High-Productivity Software Design in the Multi/Many-core Era

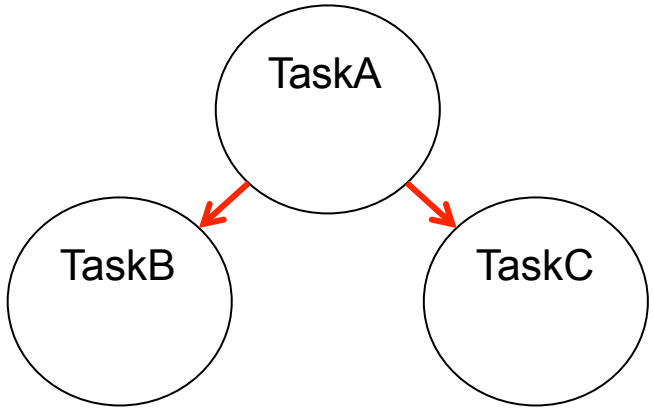


High-Productivity Software Stack for Multi/Many-core Systems



Topic 1: Task based programming models

Task-based Dataflow Prog. Models



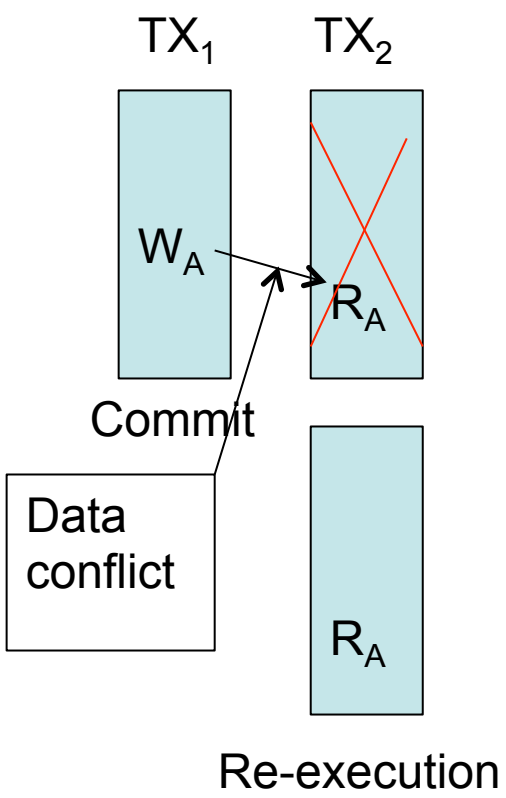

```
#pragma css task output(a)
void TaskA( float a[M][M]);

#pragma css task input(a)
void TaskB( float a[M][M]);

#pragma css task input(a)
void TaskC( float a[M][M]);
```

- Programmer annotations for task dependences
- Annotations used by run-time for scheduling
- Dataflow task graph constructed dynamially

Hypothesis: Programmers focus on extracting parallelism, system delivers performance. BUT: Is this a good idea?

Topic 1:
Transactional memory

Transactional Memory (TM)



- **Transactional memory semantics:**
  - Atomicity, consistency, and isolation
  - Tx_begin/Tx_end primitives
- Allow for concurrency inside critical sections
- Software implementations too slow
- Hardware implementations complex but have been adopted (IBM Bluegene, Intel Haswell)
- 100s of papers in the open literature; design space fairly well understood

Hypothesis: Simplifies for programmers, but is this a good idea?