Programmability of multi-core architectures

Jos van Eijndhoven

CTO & Co-founder

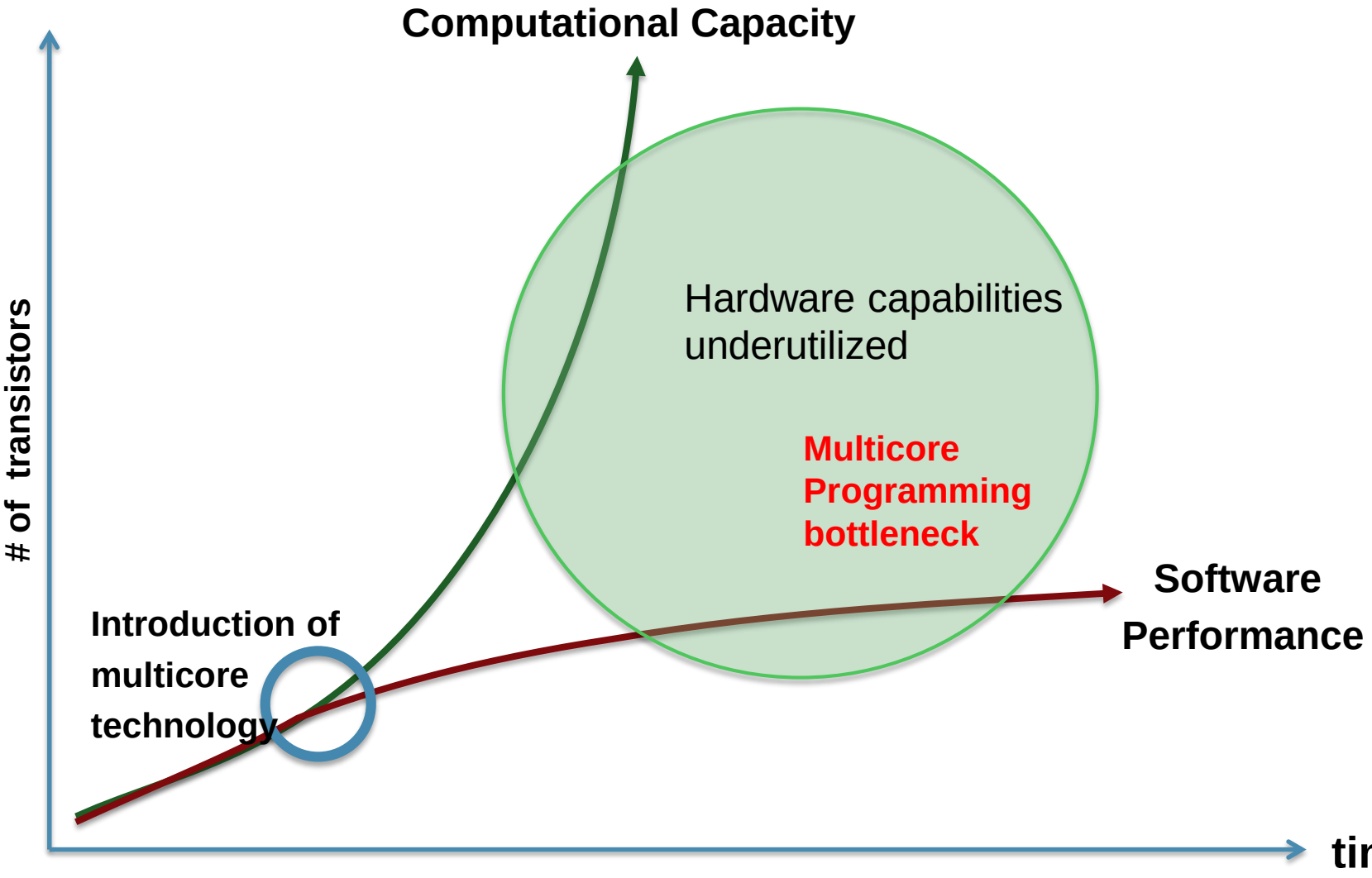
MPSoC 2014

Margaux, France, July 2014

jos@vectorfabrics.com

Eindhoven, The Netherlands

Moore's law versus Amdahl's law



What about parallel programming bugs?

Global `int x = 0, y = 0;`

Launch four threads, namely:

- Thread 1: `x = 1;`
- Thread 2: `y = 1;`
- Thread 3:

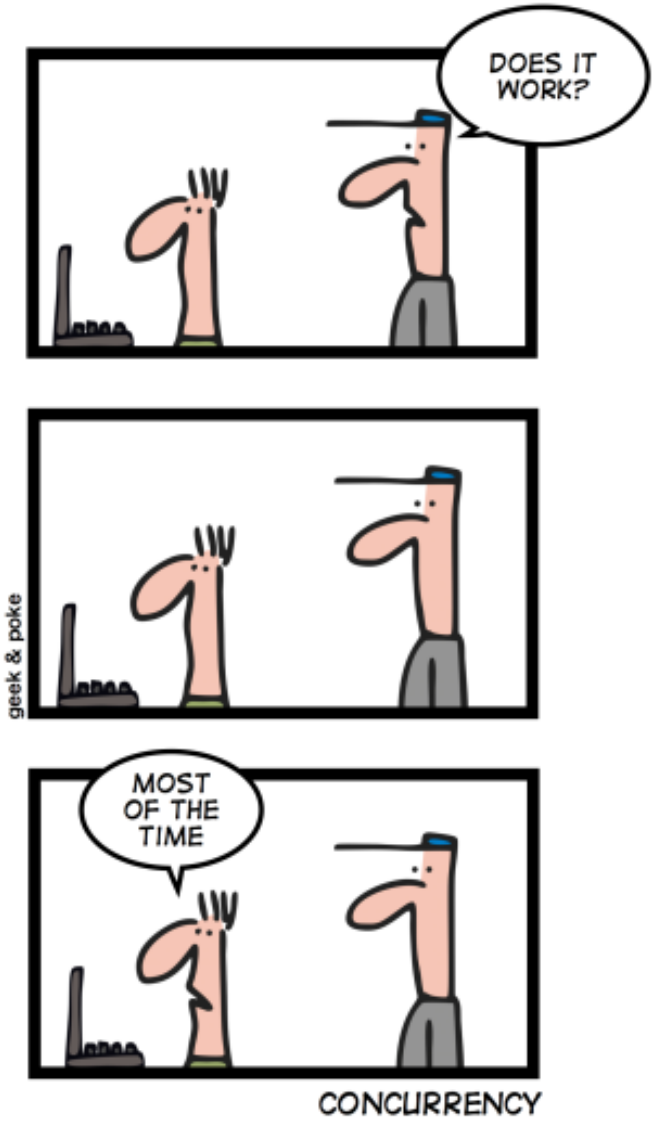
```
if (x && !y) print("X first");
```

- Thread 4:

```
if (y && !x) print("Y first");
```

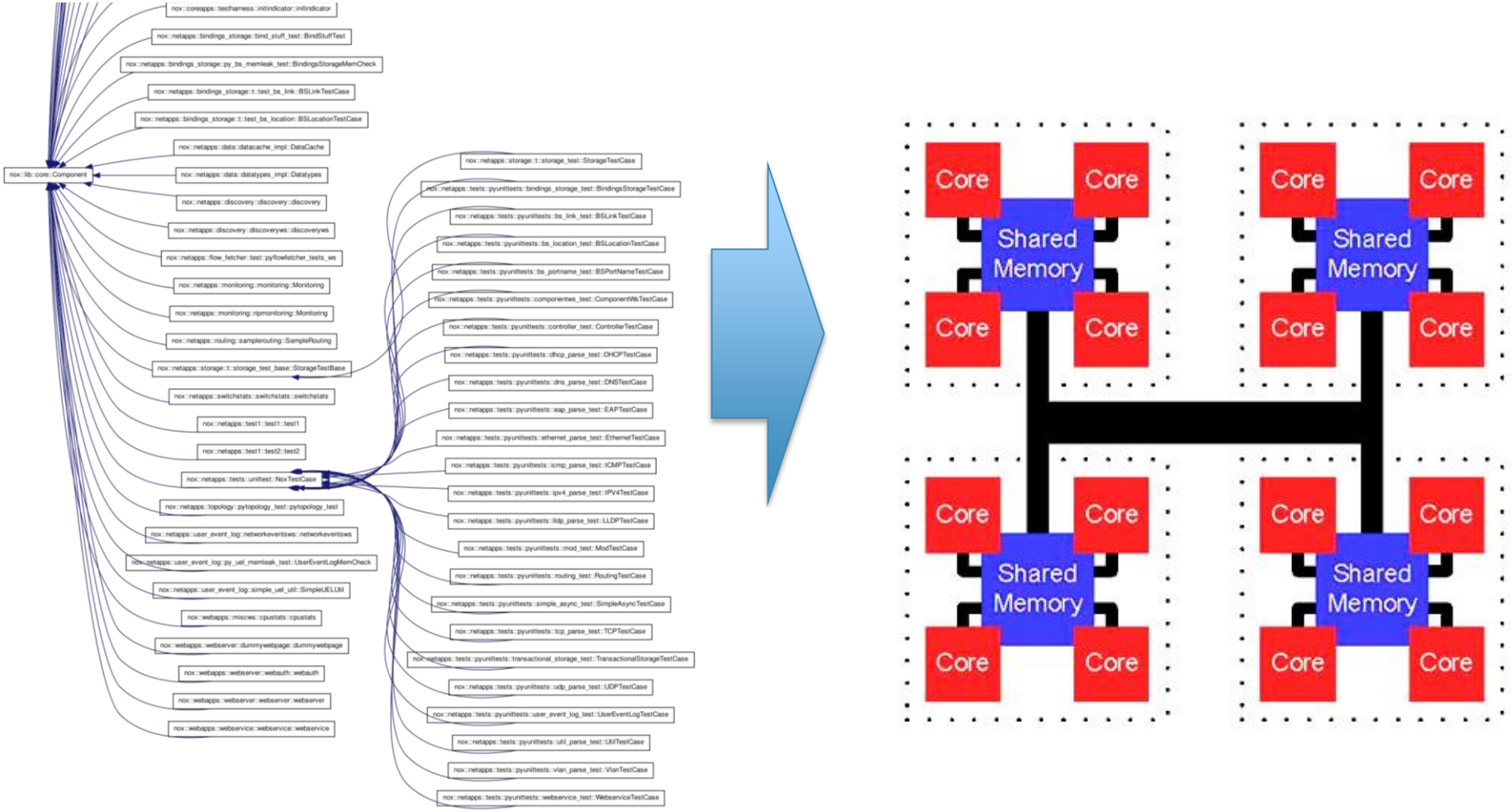
Memory ordering issues result in nondeterministic behavior!

Leads to hard-to-find bugs!



Partitioned memory?

Embedded products: growing complexity of code and data



Strong foundation – learn from history?

- Shared-memory multi-cores
- Cache-coherent
- Virtual memory paging
- Single shared OS kernel

Successful abstraction!
Proven by 30 years of collaborative compute architecture history

IBM, SUN, HP, Intel, AMD, ARM …

Be careful to create proprietary non-standard solutions!

Classic C/C++ for multi-threading

Three basic primitives, and some OS-level functionality

- Volatile variable declarations:
  force compiler load/store generation, limit compiler re-orderings
- Memory fence operations:
  force load/store ordering at runtime in the memory system
- Atomic operations:
  indivisible read-modify-write (increment, test-and-set)
- Higher-level constructs (semaphores, condition variables) that include OS and kernel support ➔ thread sleep and wakeup

Only ‘volatile’ is standardized in C/C++. Originally designed for I/O to hardware.

Posix thread library in 1995, fences/atomics are compiler specific intrinsics

Classic C/C++ constructs for threading

Three basic primitives, and some OS-level functionality

- Volatile variable declarations:
  force compiler load/store generation, omit compiler re-orderings
- Memory fence operations:
  force load/store ordering done in the memory system
- Atomic operations:
  indivisible read-modify-write (increment, test-and-set)
- Higher-level abstractions (semaphores, condition variables) that include OS and kernel support ➔ thread sleep and wakeup

Get rid of all of this 15 years of programming practice

bold move by the C++11 committee!

C11/C++11 parallel programming

Creation of multi-threaded programs:

- The C/C++ compiler will always assume multi-threaded access to variables with global scope. This inhibits some optimizations. (C++11 has no ‘volatile’ to denote inter-thread data exchange)
- Atomic operations are overloaded with memory fence behaviors. These are the basic building blocks for inter-thread synchronization.

Finally: multi-core memory behavior is specified for C/C++ !

it standardizes on a weak memory ordering!

Your MPSoC adheres to this memory ordering semantics?

Too many paradigms...

OpenCL

MPI
Message Passing Interface Standard

OpenMP

NVIDIA
CUDA

OpenACC
DIRECTIVES FOR ACCELERATORS

THE Multicore ASSOCIATION
MCAPI

RenderScript

Intel
Cilk™ Plus

Semiconductor vendor's sales challenge

Look at the raw performance and power features of our **multicore chip**!

Multicore vendor

What is the cost of **porting my code** to your multicore?

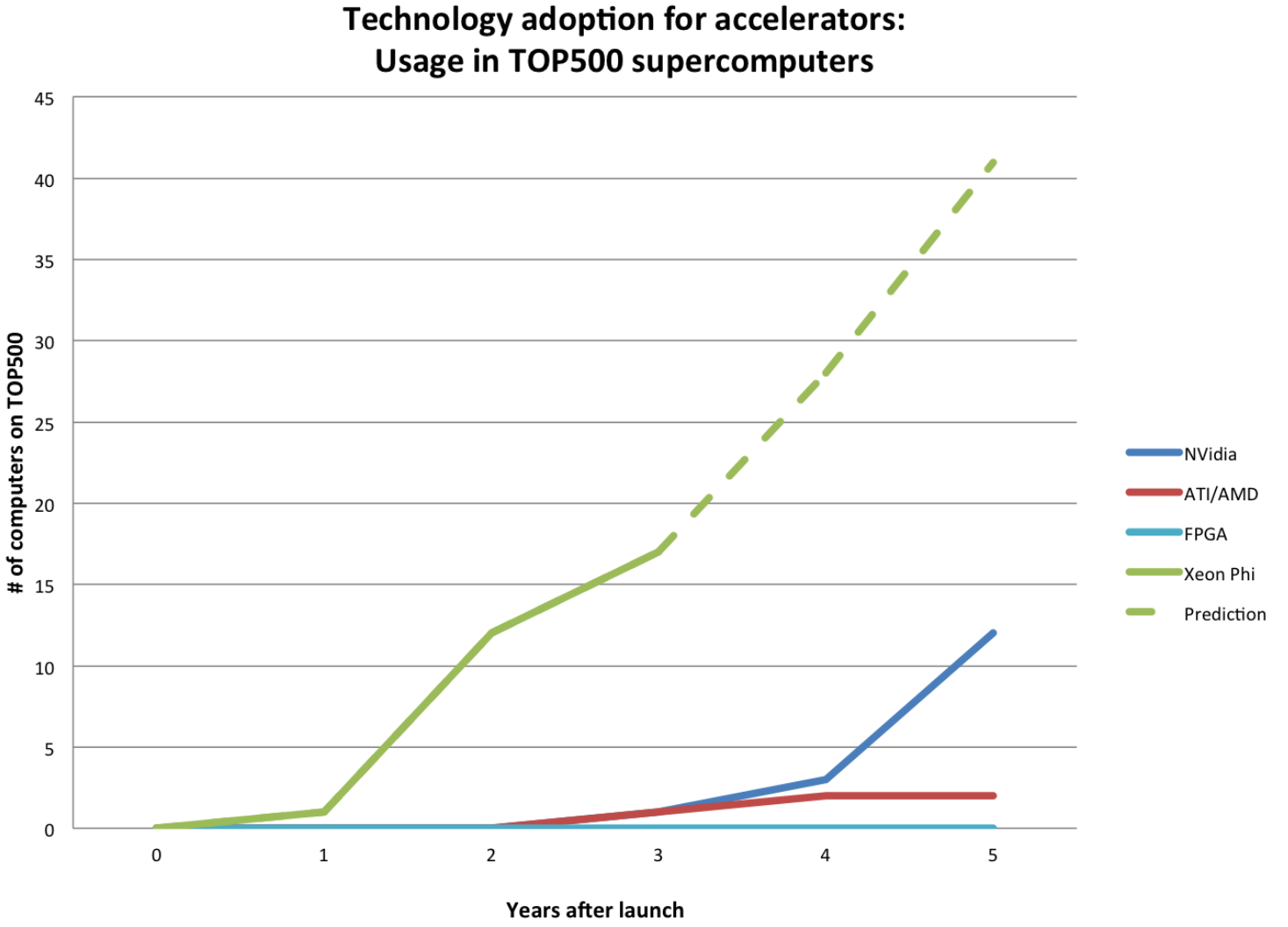
Show me what the **performance and power** of **my code** will be on your multicore

I need **support**! I don't understand how to benefit from the hardware features through my code

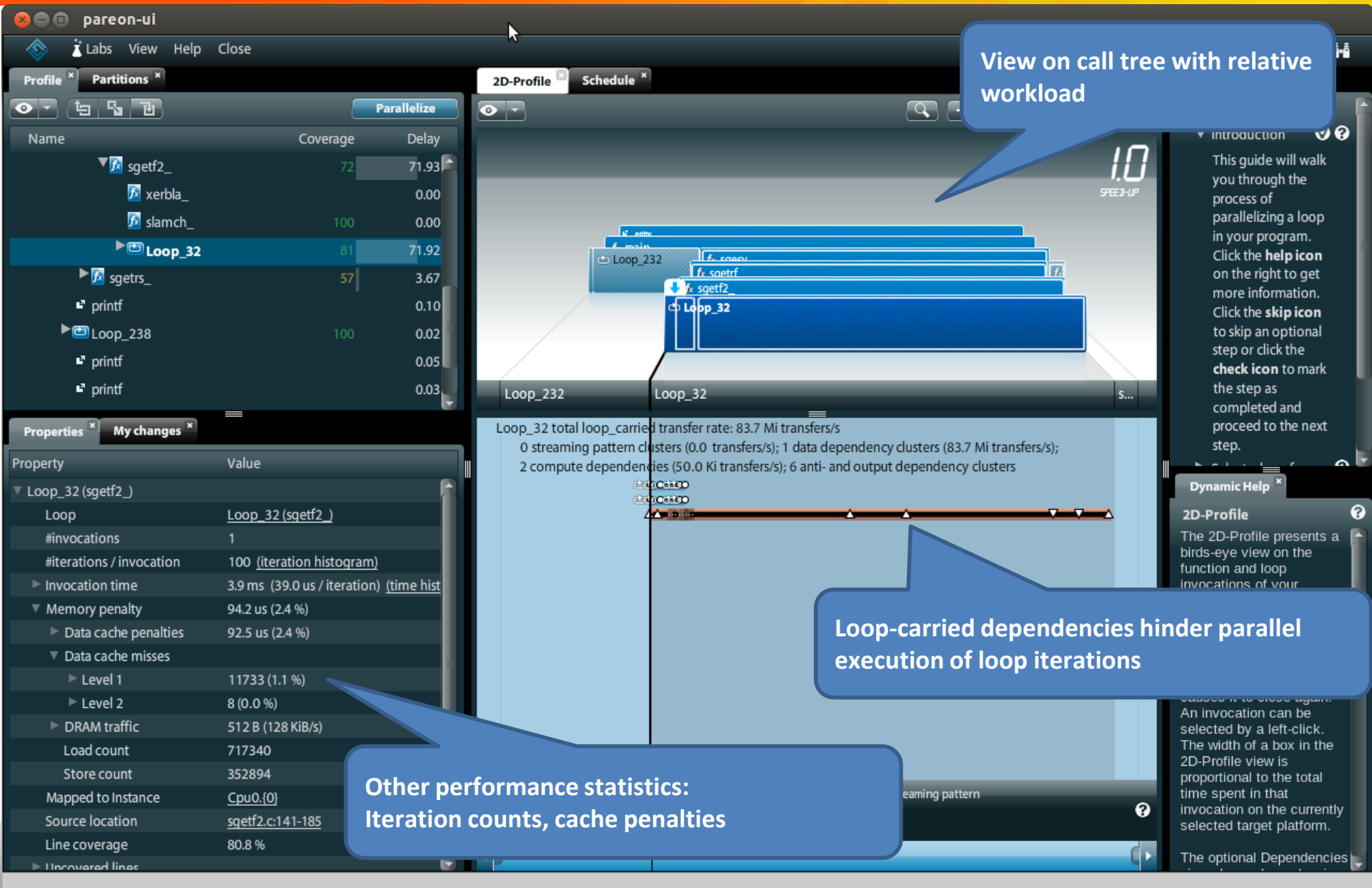
Device OEM

Good programmability is required to win enough customers!

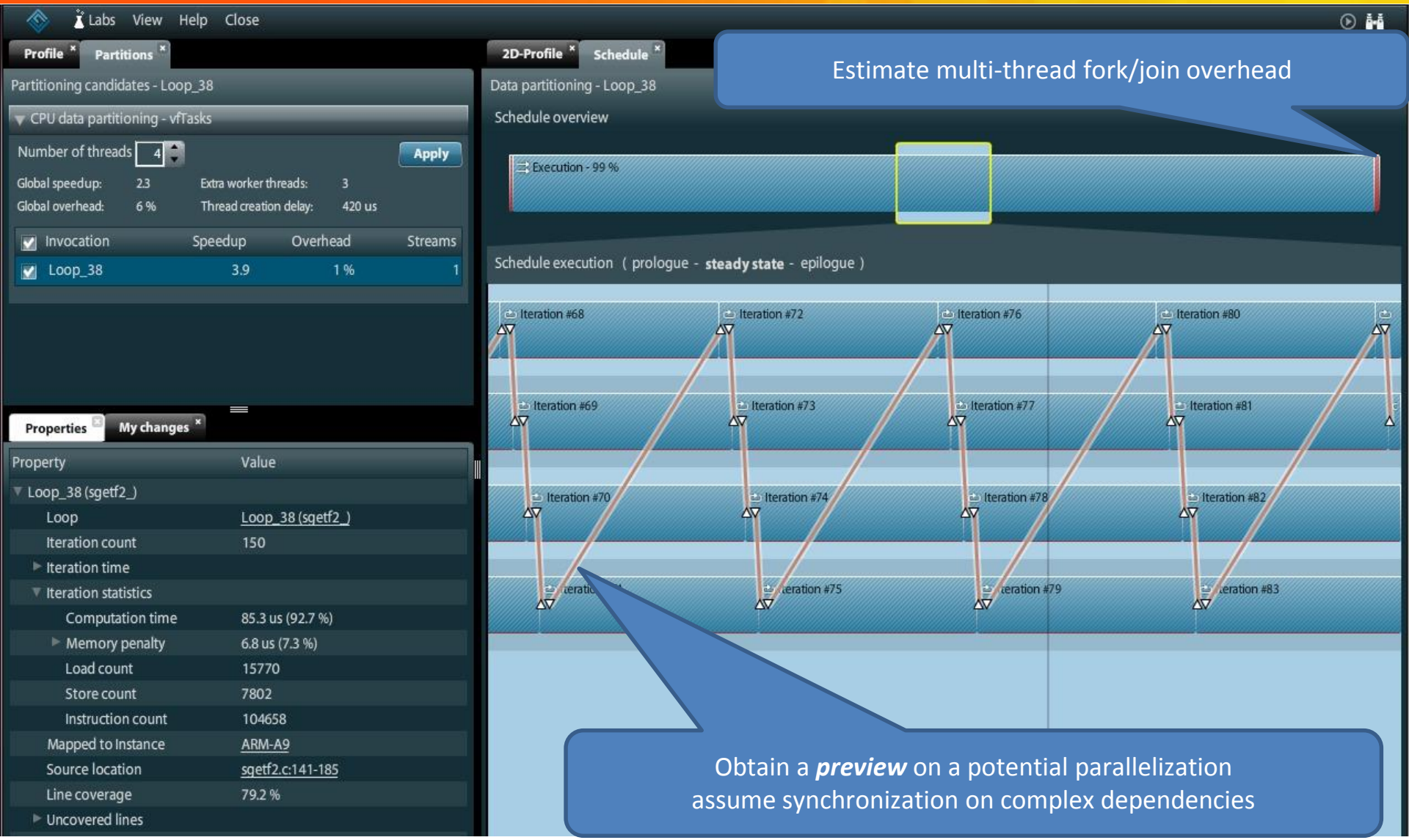
Value of programmability: fast adoption



PAREON: performance analysis



PAREON: Schedule data dependencies



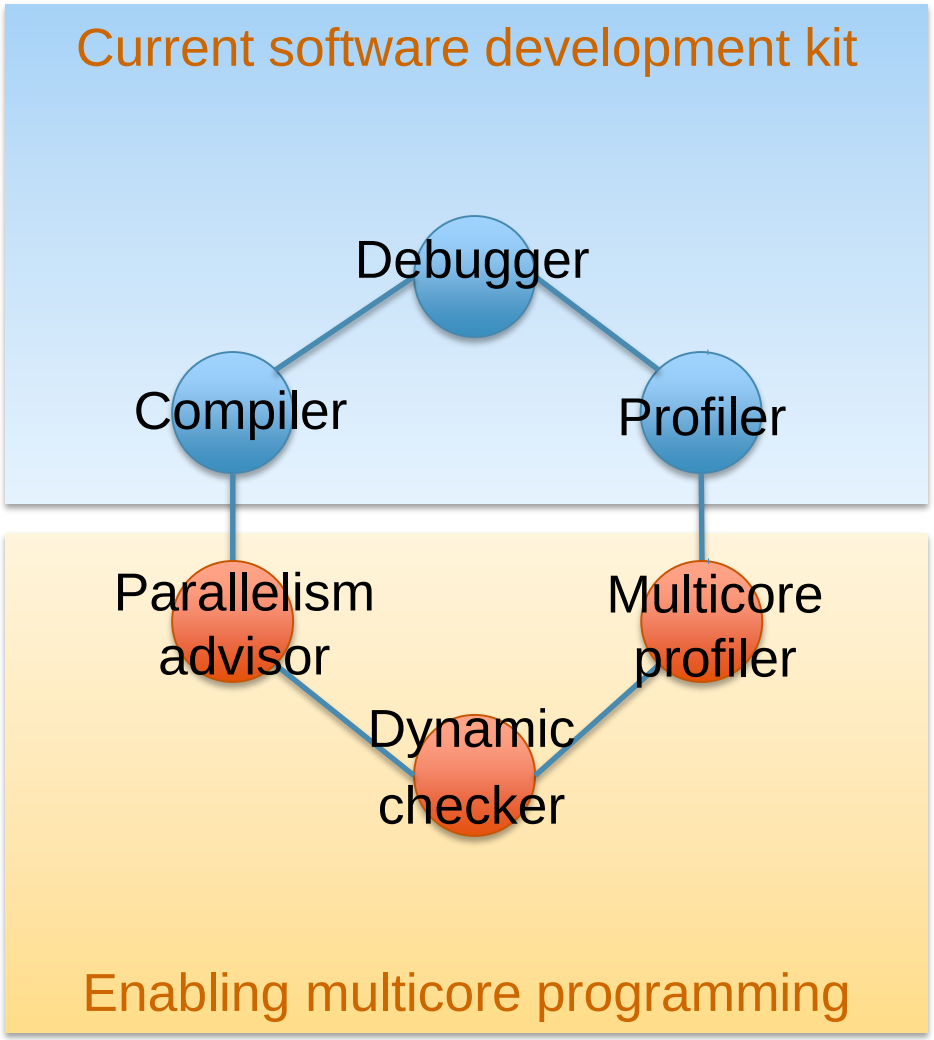
Pareon technology

- Compile C/C++ sources with instrumenting compiler
- Hijack calls to OS and standard libraries through runtime binary rewrite
- Application execution creates event trace
- Analyze event trace for effect on (some other) target platform

Works for real-world code:

- Google Chromium browser: 10M+ lines of C++, parallelized loop that contains 1M+ lines of code.
- Game engine with 500Klines of code

Software development kit – more tools?



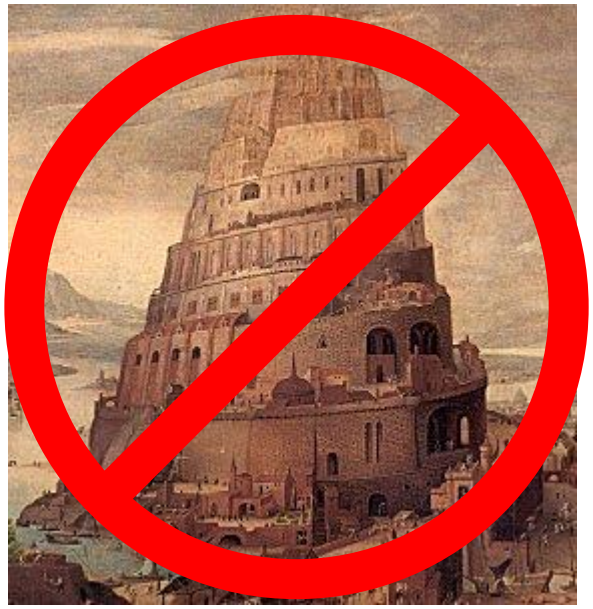
- Basic SDK focuses on programming on just one core
- Multicore tools needed for interactions between cores

→ **Parallelism advisor** – insight in how to partition and map code & data structures

→ **Multicore profiler** – optimize scheduling and memory accesses

→ **Dynamic checker** – check for multicore errors such as data races, deadlocks

Takeaway



1. Multicore programming challenge hampers adoption of platforms
2. There is no unified manycore programming paradigm, only partial solutions
3. Programmers need further multicore programming tools in their SDK

Create multicore chips for programmers,
not for hardware designers

Optimize your software

http://vectorfabrics.com

Jos van Eijndhoven

jos@vectorfabrics.com

+31 40 8200960

Eindhoven, The Netherlands

Vector Fabrics