



Salishan conference, April 2009

# Impacts of Energy Efficiency on Supercomputer Programming Models

Craig Stunkel, IBM Research

# What is a programming model?

- **A programming model is a **story****
  - A common conceptual framework
  - Used by application developers, algorithm designers, compiler-writers, runtime developers, tool builders to communicate with each other and write code.
  - A **good** programming model is a **robust story**
    - Makes sense to all stakeholders
    - Meets a critical need
- **May be realized through one or more of:**
  - Libraries
  - Language/compiler extensions – pragmas, directives
  - New languages
- **Different programming models may exist at different levels of abstraction**
- **A good programming model can lead to new industry-wide eco-systems**
  - E.g. Java, Map-Reduce, ...

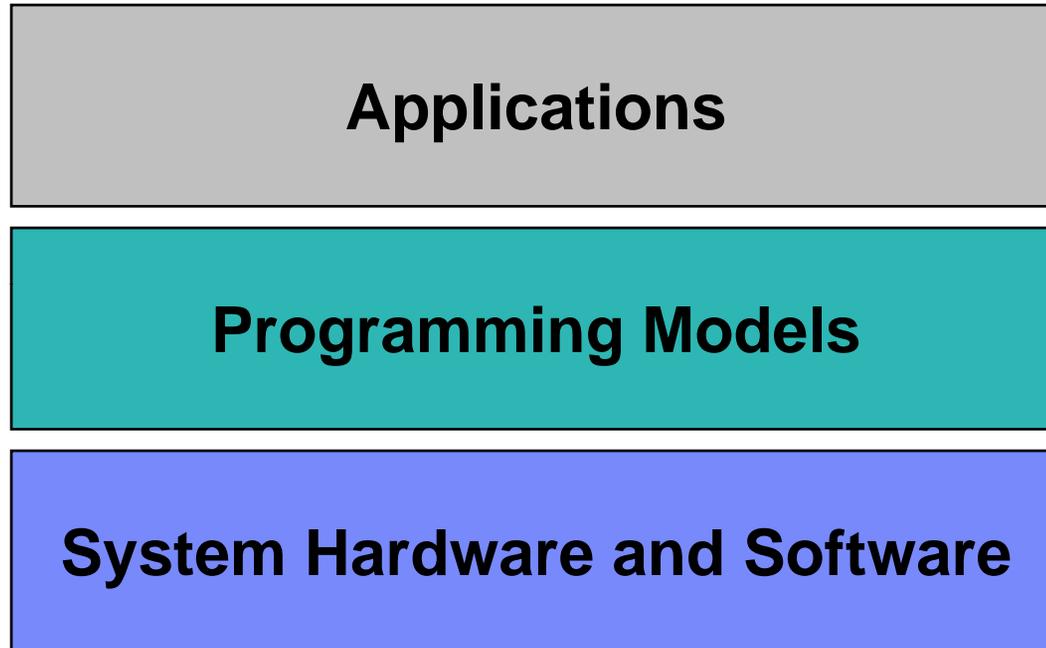
Slide courtesy of Vijay Saraswat

## Desirable programming model characteristics

- **Realizable in existing tool-chains (C, Fortran, Java, OpenMP, scripting languages) with minimal changes**
  - E.g. addition of a few directives
  - Single source code
- **Should be performance portable across architectures**
- **Should cover a sweet spot of applications**
  - i.e. do really well on targeted workloads on targeted architectures
- **Should support source-level performance debugging (tools)**
- **Should provide smooth performance vs effort graph**
  - With low startup cost
- **Should mesh well with scale-out programming model**
  - Ideal: single unified programming model from accelerators to clusters

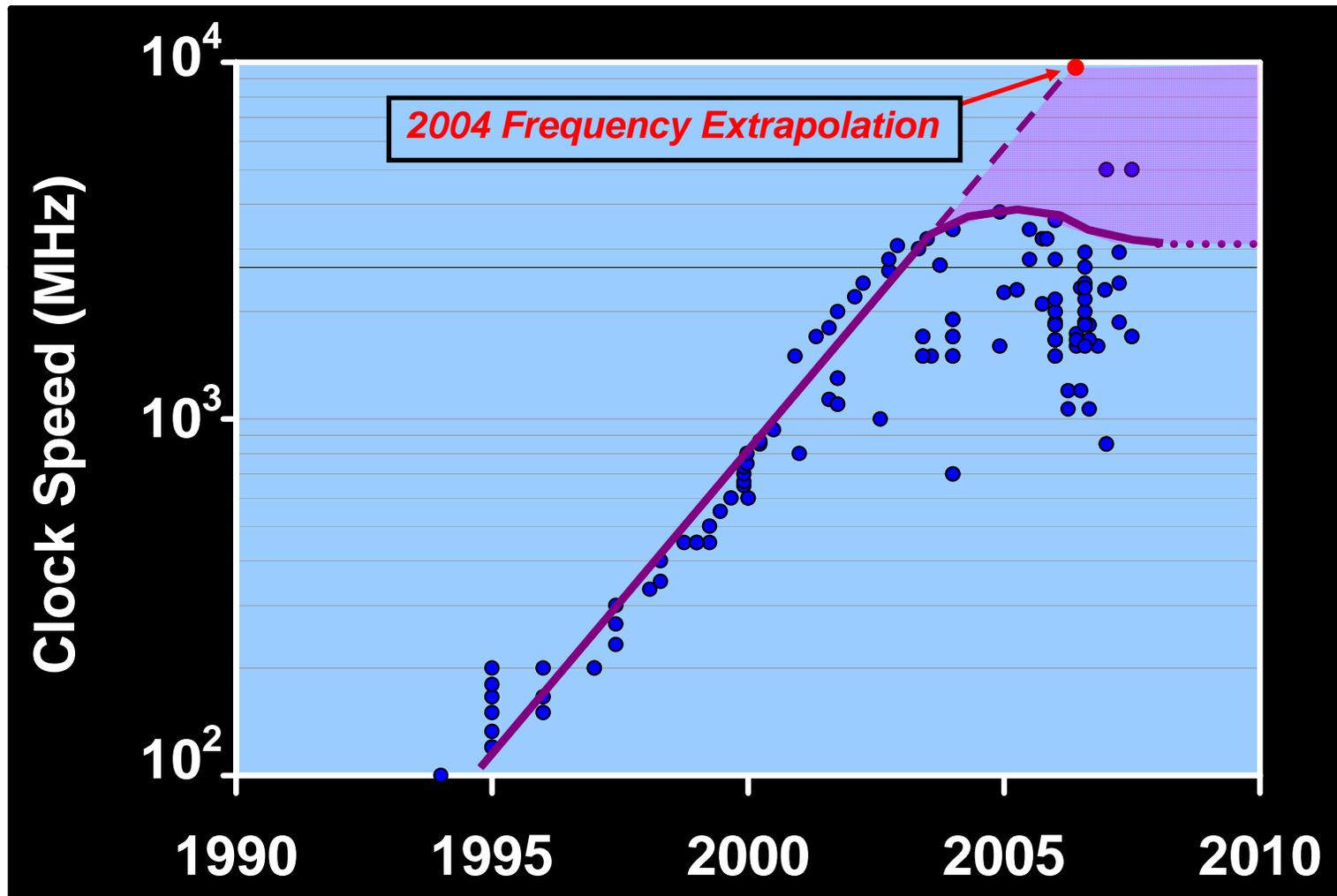
Adapted from Vijay Saraswat

# Programming models



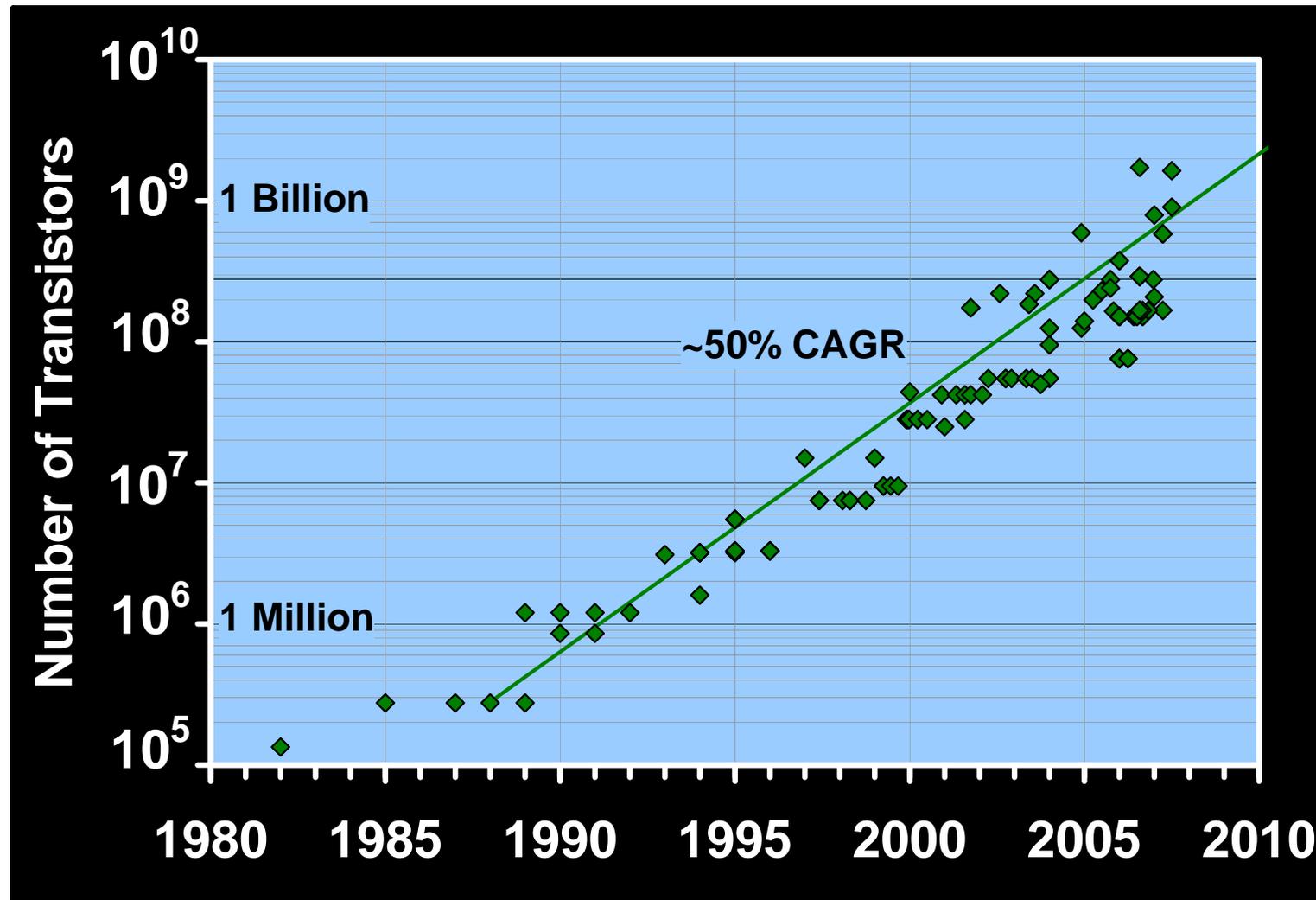
# Microprocessor Clock Speed Trends

Managing power dissipation is limiting clock speed increases



# Microprocessor Transistor Trend

**Moore's (original) Law alive: transistors still increasing exponentially**

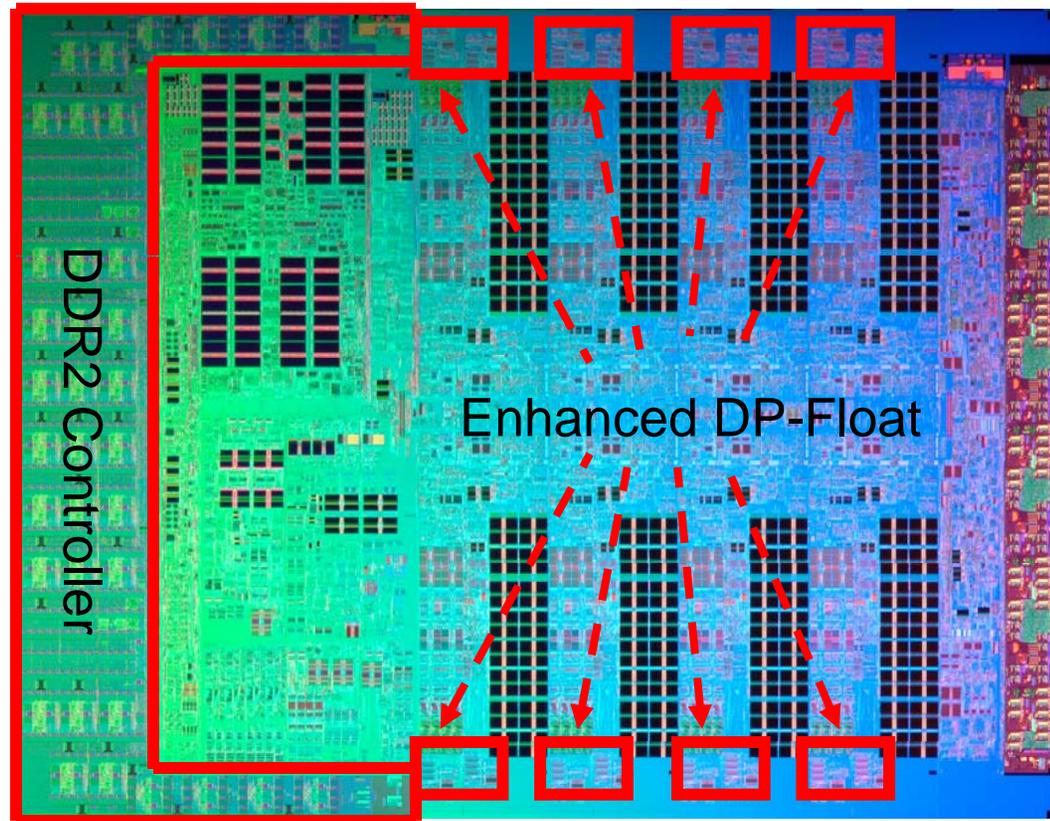


## Hardware trends that address the power problem

- **Trend #1: Multicore processor chips**
  - Maintain (or even reduce) frequency while replicating cores
  
- **Trend #2: Accelerators**
  - Previously, processors would “catch” up with accelerator function in the next generation
    - Accelerator design expense not amortized well
  - **New accelerator designs more likely to maintain performance advantage**
    - And will maintain an enormous power advantage for target workloads

# The IBM PowerXCell 8i Processor

- **Implementation of Cell Broadband Engine Architecture**
- **Fully pipelined double precision FP**
- **DDR2 SDRAM support**
  - Up to 16 GB / chip
- **Speeds & Feeds**
  - 108.8 DP FLOPS
  - 217.6 SP FLOPS
  - 25.6 GB/s mem B/W



## Hardware trends that address the power problem

- **Trend #2b: Heterogeneous multicore in general**
  - Mixes of powerful cores, smaller cores, and accelerators potentially offer the most efficient nodes
  - The challenge is harnessing them efficiently

See “Amdahl’s Law in the Multicore Era” by Mark Hill

## Other hardware trends

- **Tighter integration of memory**
  - Improves both power and performance
  - But storage is growing further away
  
- **Integration of optics**
  - Improves both power and bandwidth
  
- **Intrinsically less reliable**
  - Must attack via a multitude of techniques
    - May also affect programs and programming models

## Programming issues

- **Many cores per node, and accelerators/heterogeneity**
- **Future performance gains will come via parallelism (not clock speed)**
  - An unwelcome situation for HPC apps!
- **Need new programming models to exploit**
- **At the system/cluster level:**
  - Message-passing to connect node-level languages, or
  - Global addressing to make communication implicit?

# OpenCL

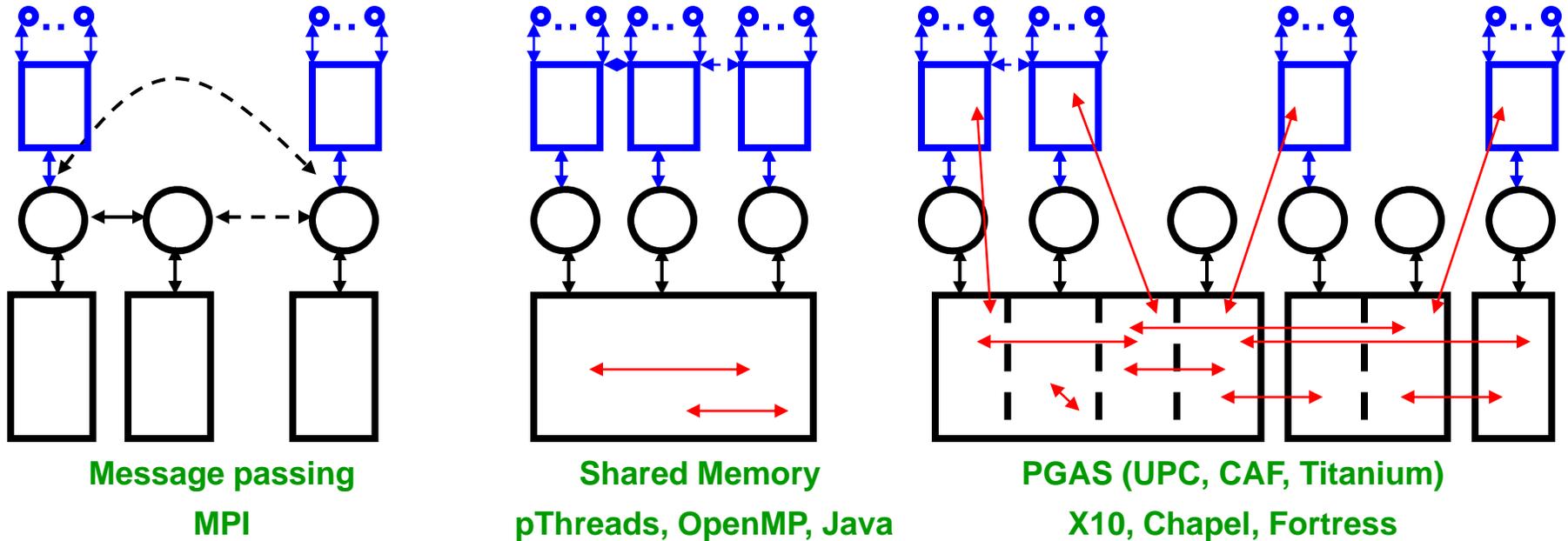
- **New open standard that specifically addresses parallel compute accelerators**
- **Extension to C**
- **Provides data parallel and task parallel models**
- **Facilitates natural transition from the growing number of (proprietary) CUDA programs**
- **Porting of Cell applications to a standard model**
- **Play wells with MPI**
  - MPI on the host for inter-node communication
- **Can interoperate with Fortran and OpenMP on the “host”**

# OpenCL

- Kernel program runs on the accelerator
- Example kernel for vector add ( $c[*] = a[*] + b[*]$ ):

```
__kernel void vec_add(__global const float *a,  
                    __global const float *b,  
                    __global const float *c)  
{  
    int gid = get_global_id(0);  
    c[gid] = a[gid] + b[gid];  
}
```

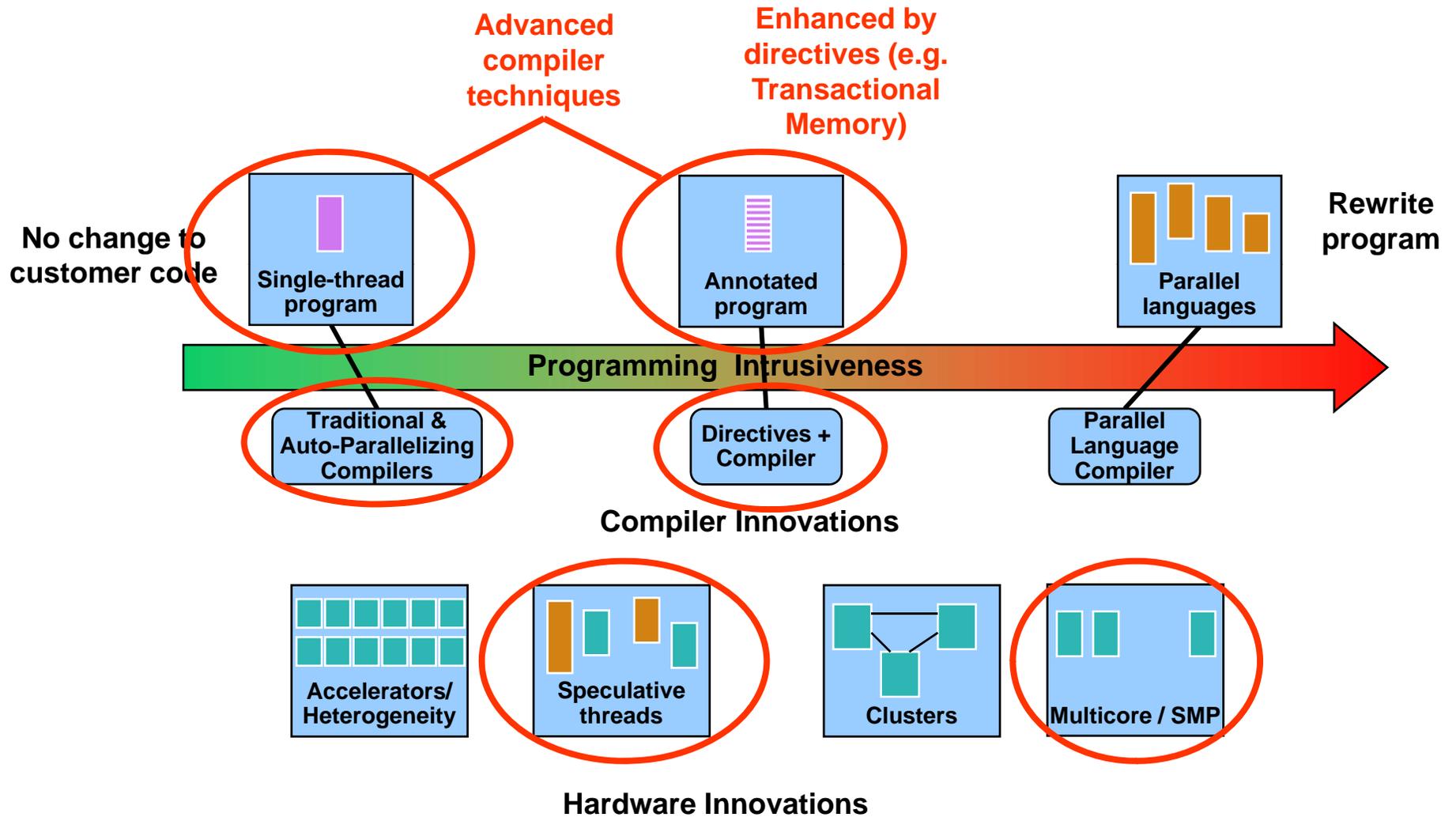
# Partitioned Global Address Space (PGAS)



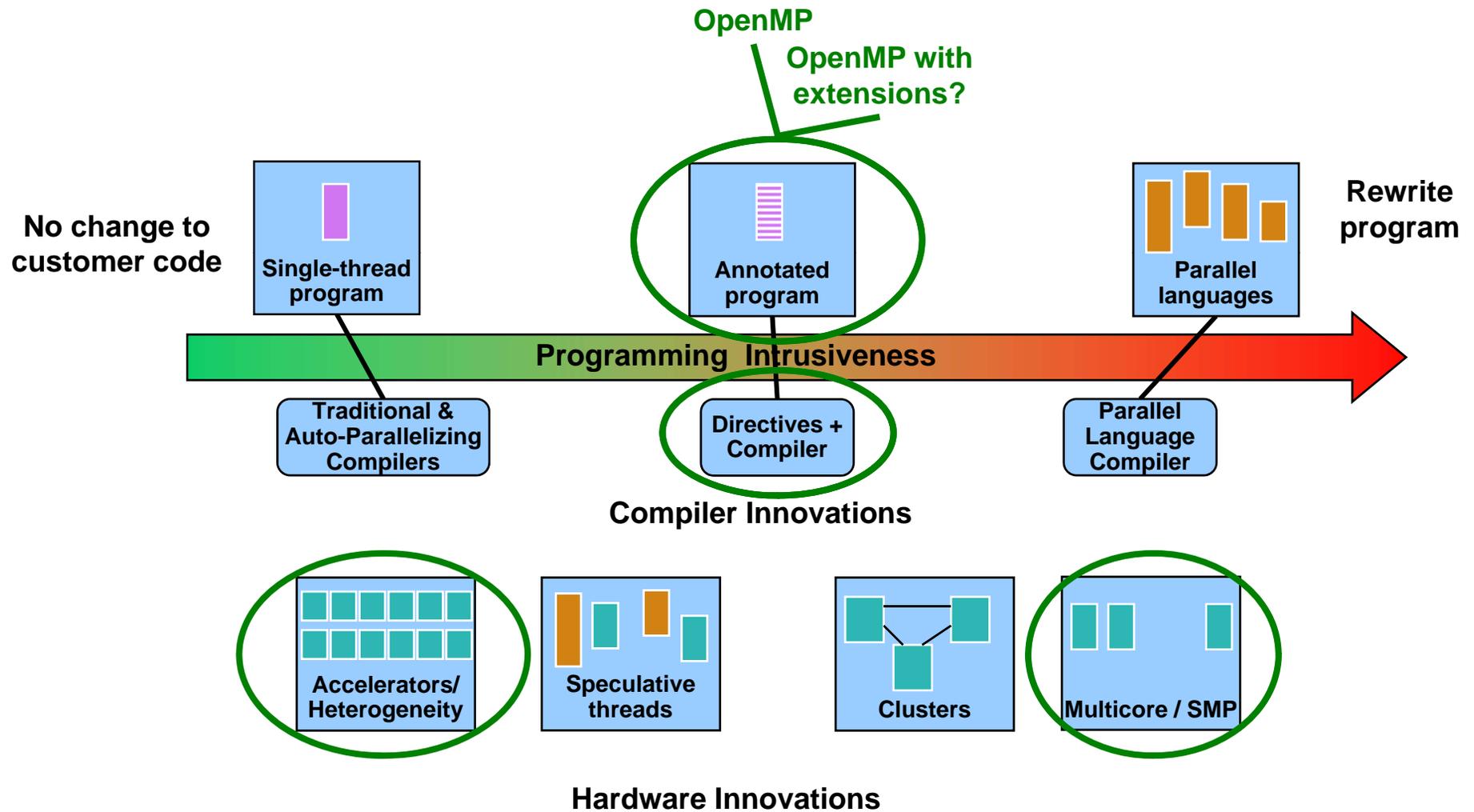
- Computation is performed in multiple **places**.
- A place contains data that can be operated on remotely.
- Data lives in the place it was created, for its lifetime.

- A datum in one place may reference a datum in another place.
- Data-structures (e.g. arrays) may be distributed across many places.
- Places may have different computational properties

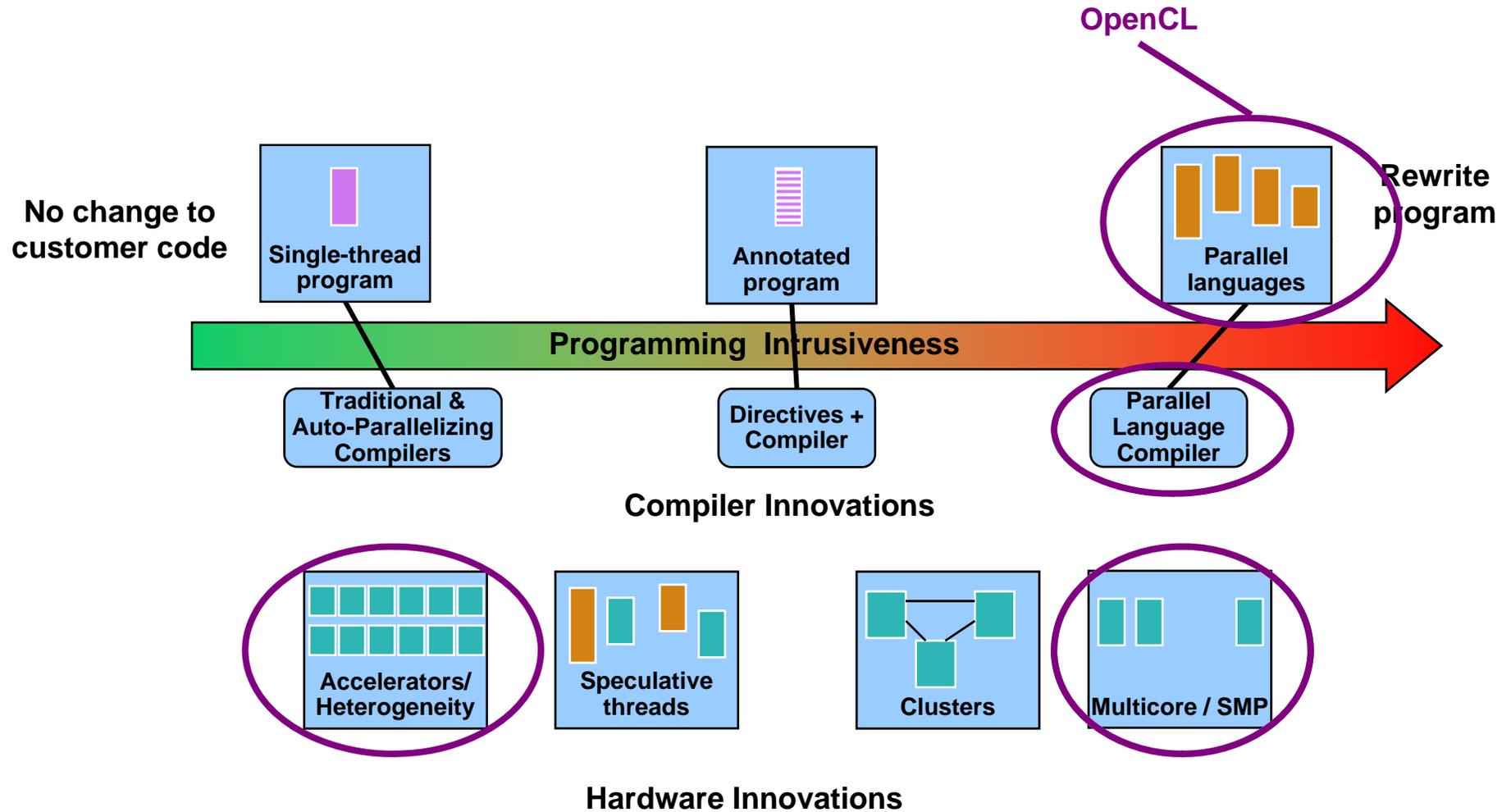
# Different approaches to exploit parallelism



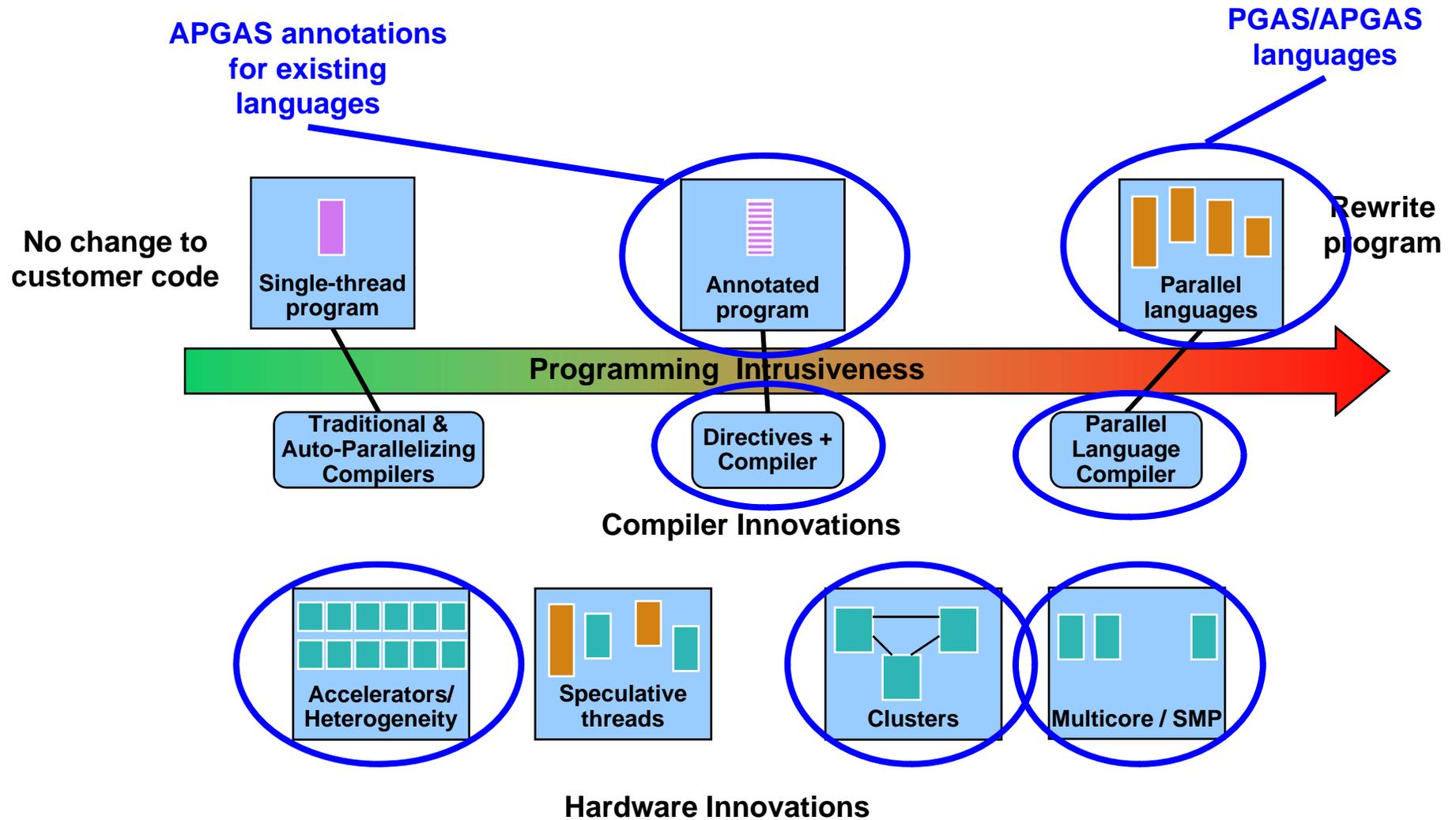
# Different approaches to exploit parallelism



# Different approaches to exploit parallelism



# Different approaches to exploit parallelism



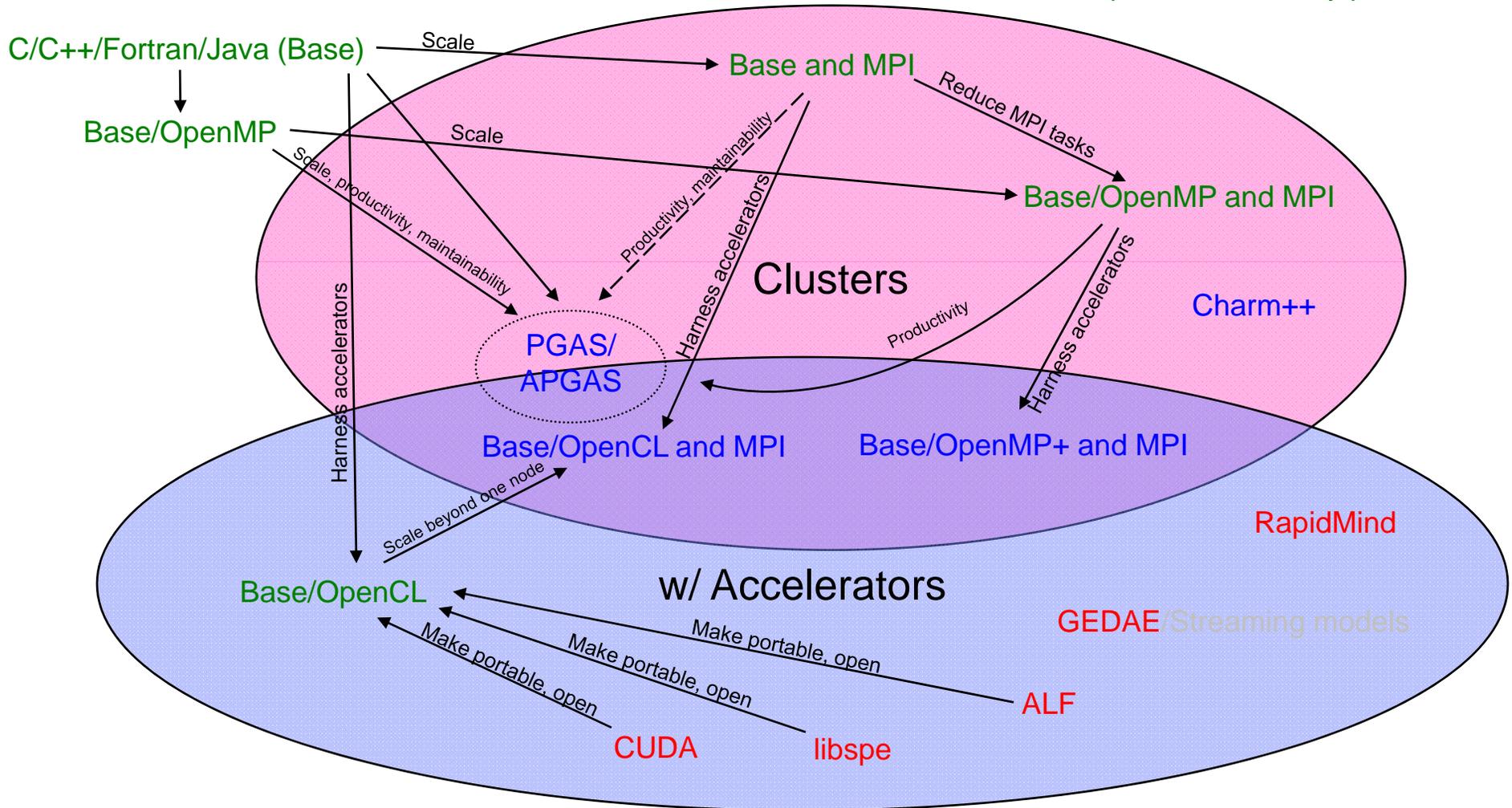
# Potential Migration Paths

Green: open, widely available\*

Blue: somewhere in between

Red: proprietary

\*OpenCL availability predicted



## Programming model perspective

- **There will be a variety of programming models**
  - No silver bullet
  - Must extend dominant legacy tool chains
    - Compilers, performance tools, ...
  
- **In particular, the industry should explore:**
  - OpenMP and potential extensions for hybrid
  - OpenCL (for compute accelerators)
  - PGAS and APGAS languages (UPC, CAF, X10, ...)
  - Pursue APGAS directives for current languages
    - C/C++, Fortran, Java

# Education

- **Producing scaling codes will be a challenge**
  
- **Too few software engineers understand how to take advantage of parallelism, particularly data parallelism**
  - Continuation of the multicore revolution is at stake
  - And supercomputers are now dependent upon multicore
  
- **How to change this?**
  - Update skills through internal and industry courses
  - Introduce parallelism and concurrency early in undergraduate programs
    - But how early (First course? Junior year?)

## Concluding thoughts

- **Multicore and heterogeneous nodes attack energy efficiency**
- **However, they introduce new programming challenges**
  - Virtually all performance gains will be due to parallelism
- **New programming models and/or languages will be required**
  - **There is no silver bullet!**
  - Adoption of new models will take time
  - Evolutionary approaches will likely prevail
- **It is time to seriously invest in PGAS**

# Questions?

