The Magnolia Programming Language

Anya Helene Bagge

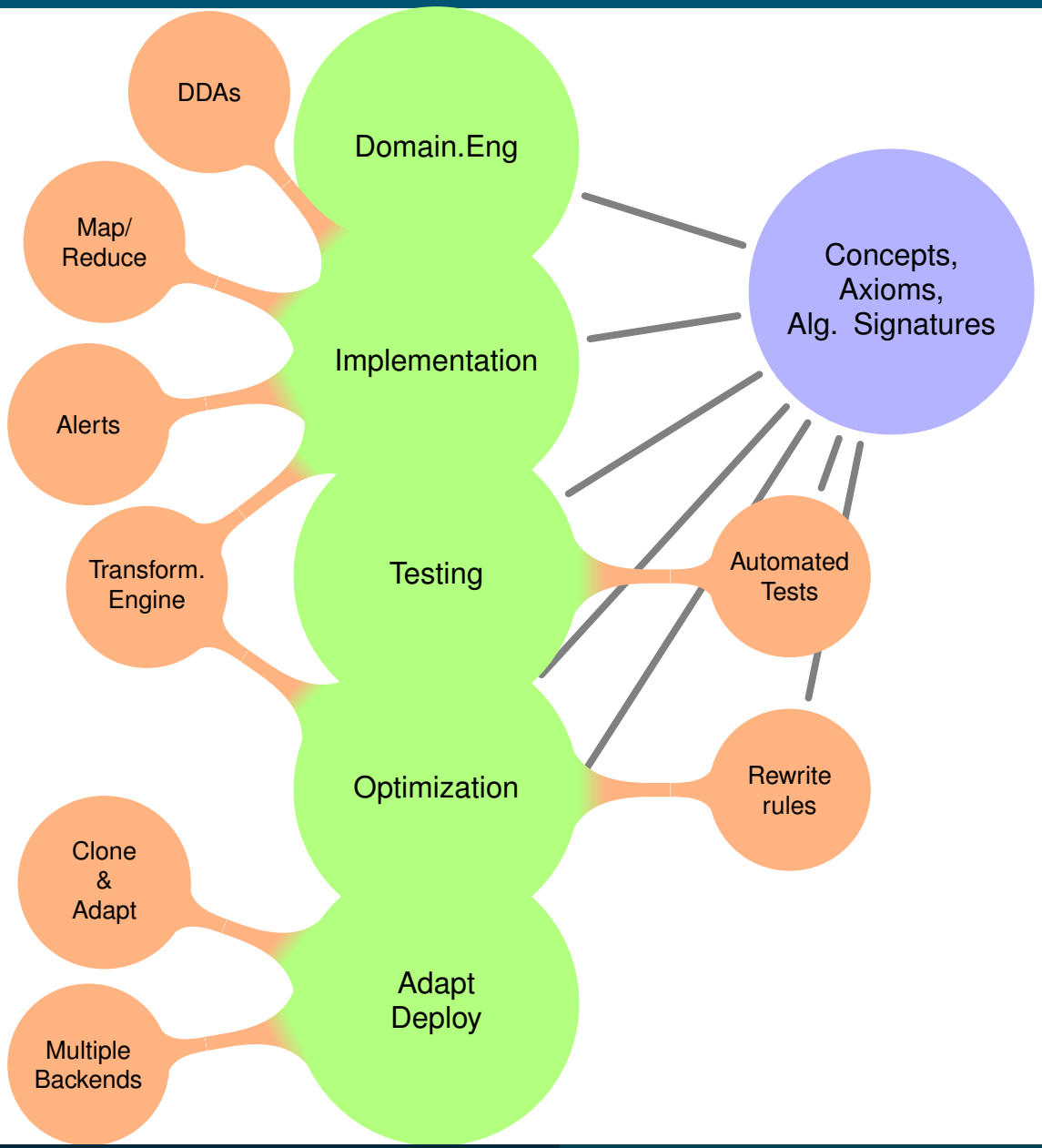
Department of Informatics
University of Bergen
Norway

PLDI'08
Student Research Competition

Introduction

- Creating scientific / HPC software is hard
  - High performance, trustworthy results, low development cost – can we get it all?
- Many projects are looking at this:
  - Fortress, X10, Chapel, C++ / Boost, ...
- We're researching a development method based on algebraic specification, abstraction and generative techniques [SAGA]
  - Sample application: Sophus, a modular PDE solver for seismic simulations
- Language support for scientific software
  - Library specific / user-defined optimization
  - Reliability: axioms, testing and error handling
  - Adaptability

Concept-Driven Development



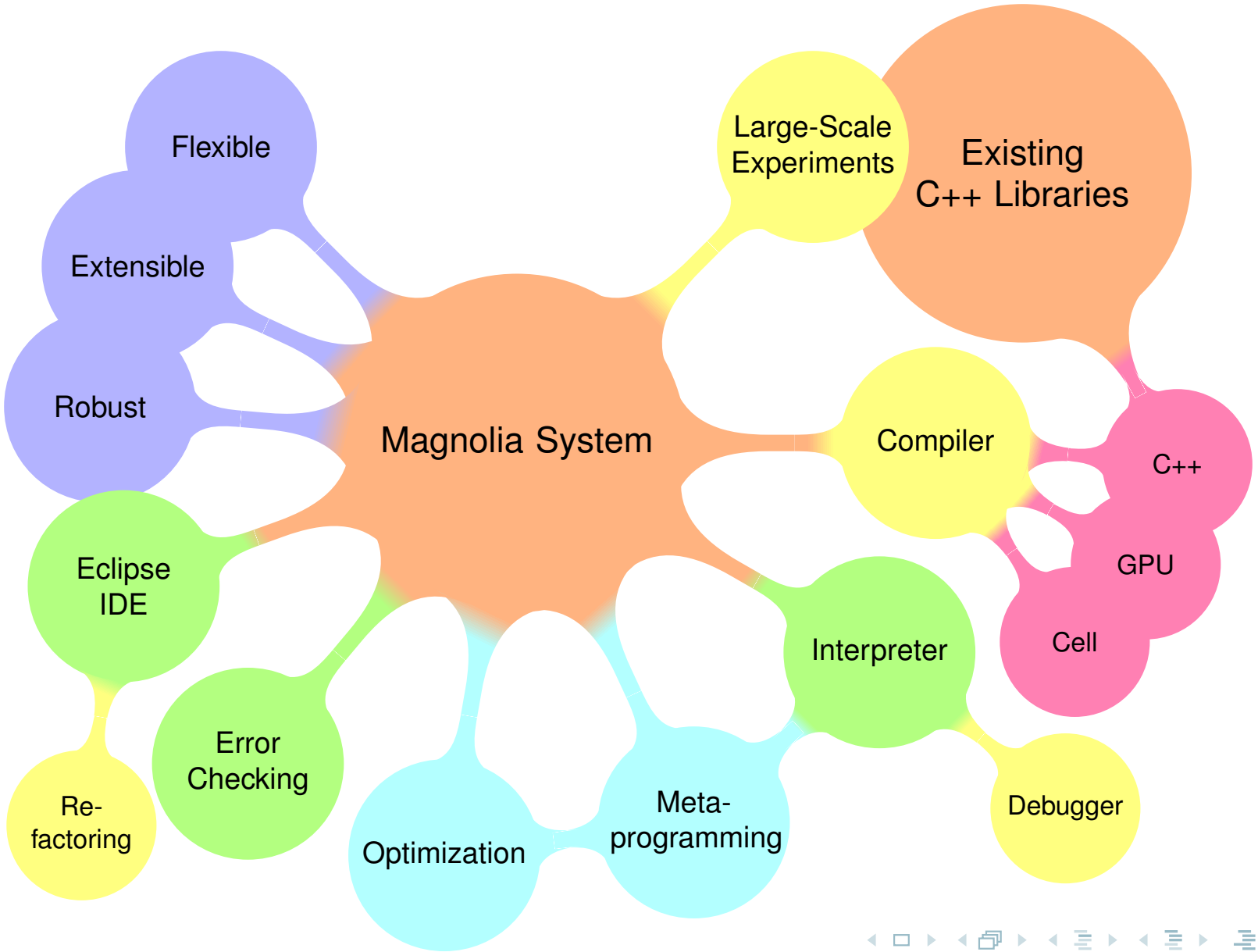

```
concept Indexable<Idx, Arr,
      Elt> {
   function Elt _[_](Arr, Idx);
   function setElt(Arr, Idx, Elt);

   axiom Indexable(Arr a,
      Idx i, Elt e)
   {
      assert setElt(a,i,e)[i] == e;
   }
}

template<Indexable T>
   procedure reverse(upd T
arr) {...}

model Indexable<int,
      Array<float>,float>;
```

Magnolia Tools and Infrastructure



Related Work

- TAMPR: Program Specification and Transformation [Boyle, et al.,89/97]
  - Our approach is more driven by axioms (and concepts)
- Generative programming
  - C++ Boost, Active Libraries, Expression Templates, Aspect-Orientation
  - Many related ideas
  - Our approach doesn't rely on ad-hoc template meta-programming – though we require a new or extended language
- 'Alert' system is inspired by CLU and Eiffel
- Axiom-based testing was introduced in the DAISTS system [Gannon, et al, 81]

Many of these ideas are quite old, but haven't seen wide use yet!

Conclusion

- Magnolia Goal: Provide a solid basis for experimentation with
  - concept/axiom-driven development method
  - novel language features; alerts, dependent typing, DDAs, ...
- Infrastructure allows rapid development of new features
  - No more wasting years trying to build prototypes for C++
- Plan:
  - Continue developing language and tool set
  - Validate design and method by applying to real-world applications
  - Incorporate research from other projects when needed

http://magnolia-lang.org/

http://www.ii.uib.no/saga/

Algebraic Signatures

We use an algebraic / functional interface to operations

- Easy to reason about, related directly to axioms, rewrite rules, specification and math
- Automatically translated to efficient imperative code

Functionalization

```
function array sort (array a);          ⟵  procedure sort (upd array a);
function array merge (array a,          ⟵  procedure merge (upd array a,
array b);                                  obs array b);
```

Mutification

```
var a = array(100);                     ⟶  array a = array(100);
var b = array(100);                        array b = array(100);
a = merge(sort(a), sort(b));               array a0 = b;
                                           call sort(a0); call sort(a);
                                           call merge(a, a0);
```