Programming Design Patterns

Patterns for High Performance Computing

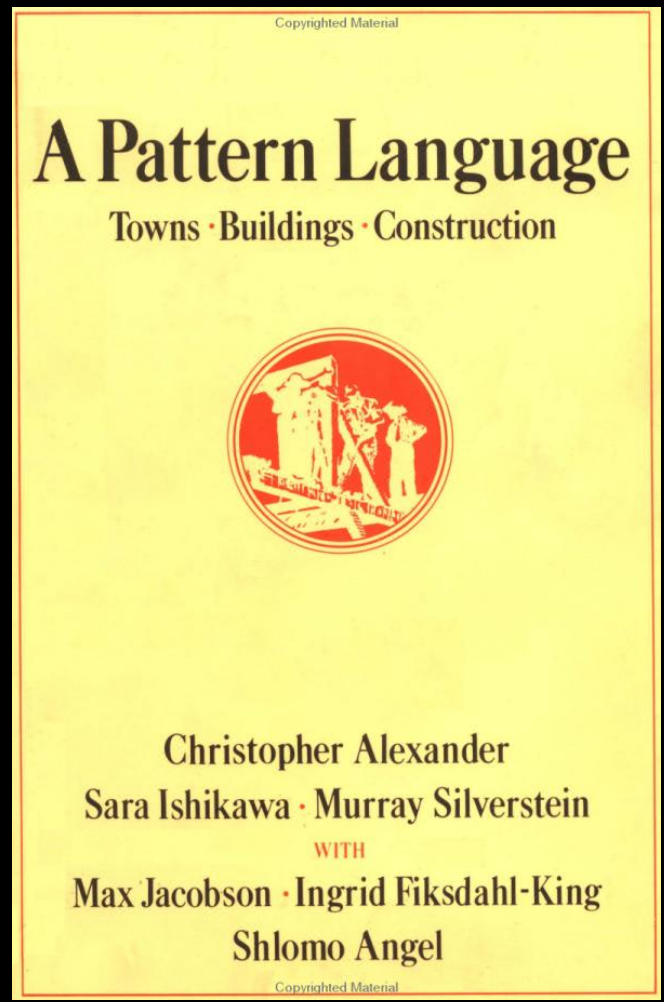
Marc Snir/Tim Mattson

Design Pattern

- High quality solution to frequently recurring problem in some domain
- Each pattern has a name, providing a vocabulary for discussing the solutions
- Written in prescribed format to allow the reader to quickly understand the solution and its context

History ‘60s and ‘70s

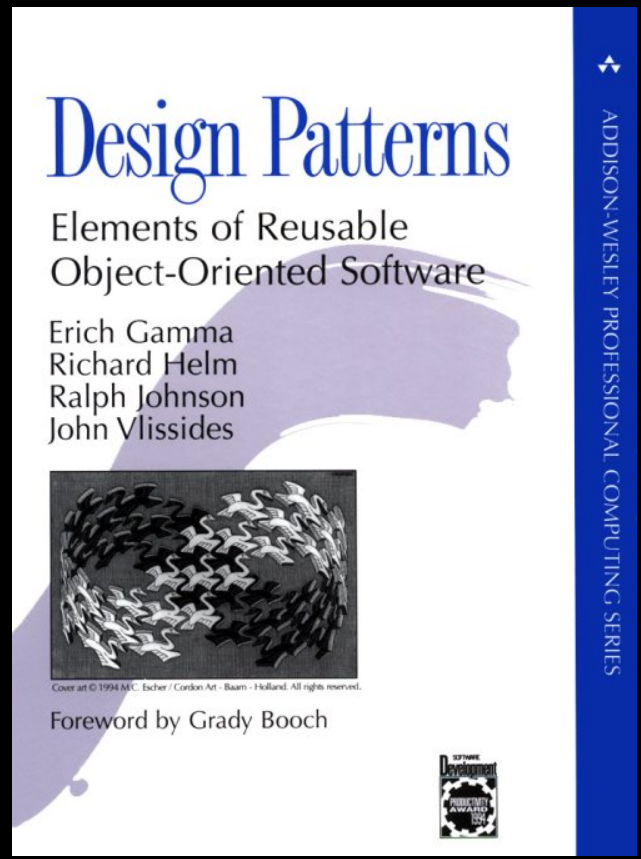
- Berkeley architecture professor Christopher Alexander
- 253 patterns for city planning, landscaping, and architecture
- Attempted to capture principles for “living” design.



Published in 1977

Patterns in Object-oriented Programming

- OOPSLA'87 Kent Beck and Ward Cunningham
- Design Patterns: Elements of Reusable Object-Oriented Software
  - By the "Gang of Four (GOF)": Gamma, Helm, Johnson, Vlissides
    - Catalog of patterns
    - Creation, structural, behavioral



Published in 1995

Impact of GOF book

- Good solutions to frequently recurring problems
- Pattern catalog
- Significant influence on object-oriented programming!
  - Created a new vocabulary for software designers.

The Task Parallelism Pattern

- Problem:
  - How do you exploit concurrency expressed in terms of a set of distinct tasks?
- Forces
  - Size of task – small size to balance load vs. large size to reduce scheduling overhead
  - Managing dependencies without destroying efficiency.
- Solution
  - Schedule tasks for execution with balanced load – use master worker, loop parallelism, or SPMD patterns.
  - Manage dependencies by:
    - removing them (replicating data),
    - transforming induction variables,
    - exposing reductions
    - explicitly protecting (shared data pattern).
- Intrusion of shared memory model…

Pattern Languages: A new approach to design

- Not just a collection of patterns, but a **pattern language:**
  - Patterns lead to other patterns creating a design as a network of patterns.
  - Patterns are hierarchical and compositional.
  - Embodies design methodology and establishes a vocabulary of design.

Patterns developed at pattern writers workshops

- Pattern languages of Programming workshops:
  - PLoP™ – since 1994 in Illinois.
  - Chili PLoP™ - since 1998 in Arizona.

  ... plus Euro PLoP™, Viking PLoP, KoalaPLoP

- Publications tend to be in books since patterns are too long to fit into typical journal papers.
  - *Concurrent programming in Java™ Design Principles and Patterns*, D. Lea
  - *Patterns for Parallel Programming*, T. Mattson, B. Sanders, B. Massingill
  - *Design Patterns C#,* S. Metsker, S. J. Metsker
  - *J2EE Design Patterns*, W. Crawford, J. Kaplan

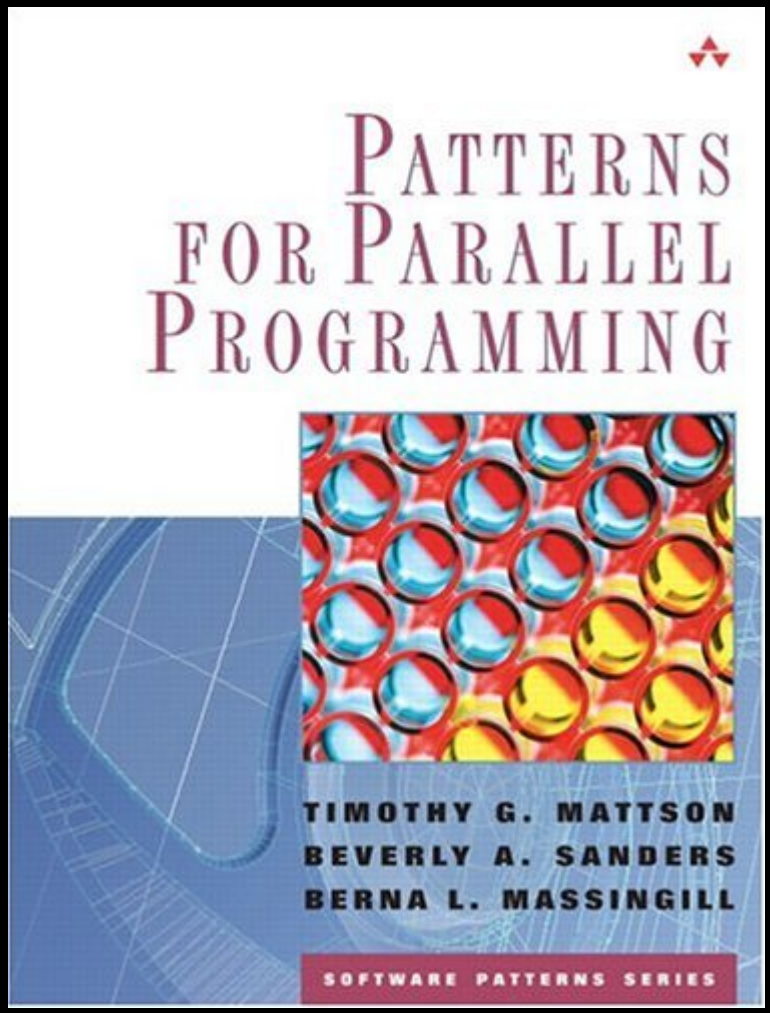
  ... and hundreds more

Pattern workshop structure

- You submit a pattern and work with a “Shepherd” to refine the pattern.
- Attend the workshop:
  - Group sits in a circle, summarizes the pattern and then discusses it.
  - Author sits outside the circle and can only listen (no talking from the author).
  - When the group is done, they invite the pattern author to join the conversation.

This can be a difficult process to sit through as a pattern author, but incredibly valuable.

Patterns for Parallel Programming

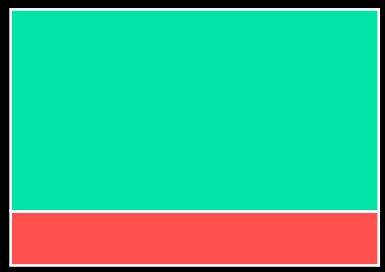


A pattern language for parallel algorithm design with examples in MPI, OpenMP and Java.

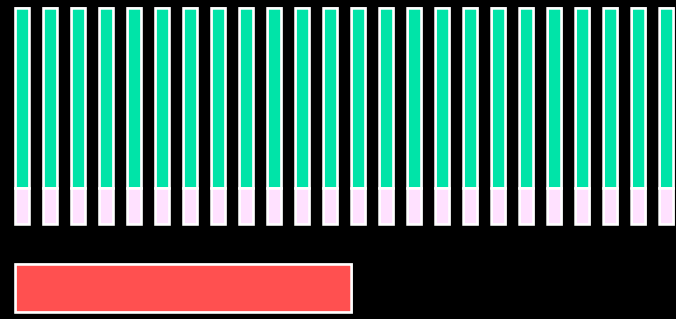
Patterns refined and developed at PLoP™ workshops from 1999 to 2003.

Our Pattern Language's structure:

Four design spaces in parallel software development



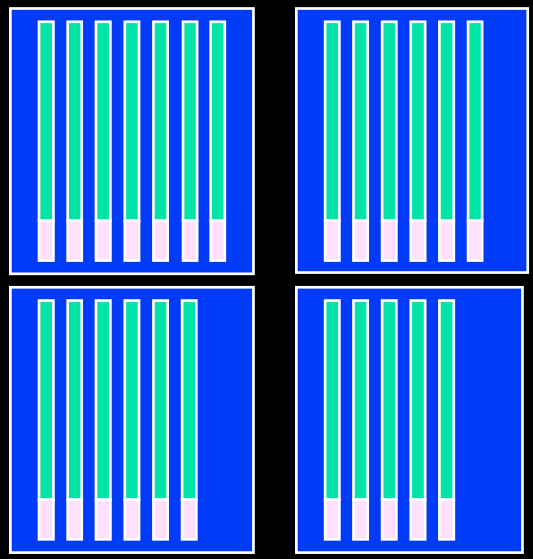
Finding Concurrency



Original Problem

Algorithm Structure

Tasks, shared and local data



Supporting struct. & impl. mech.

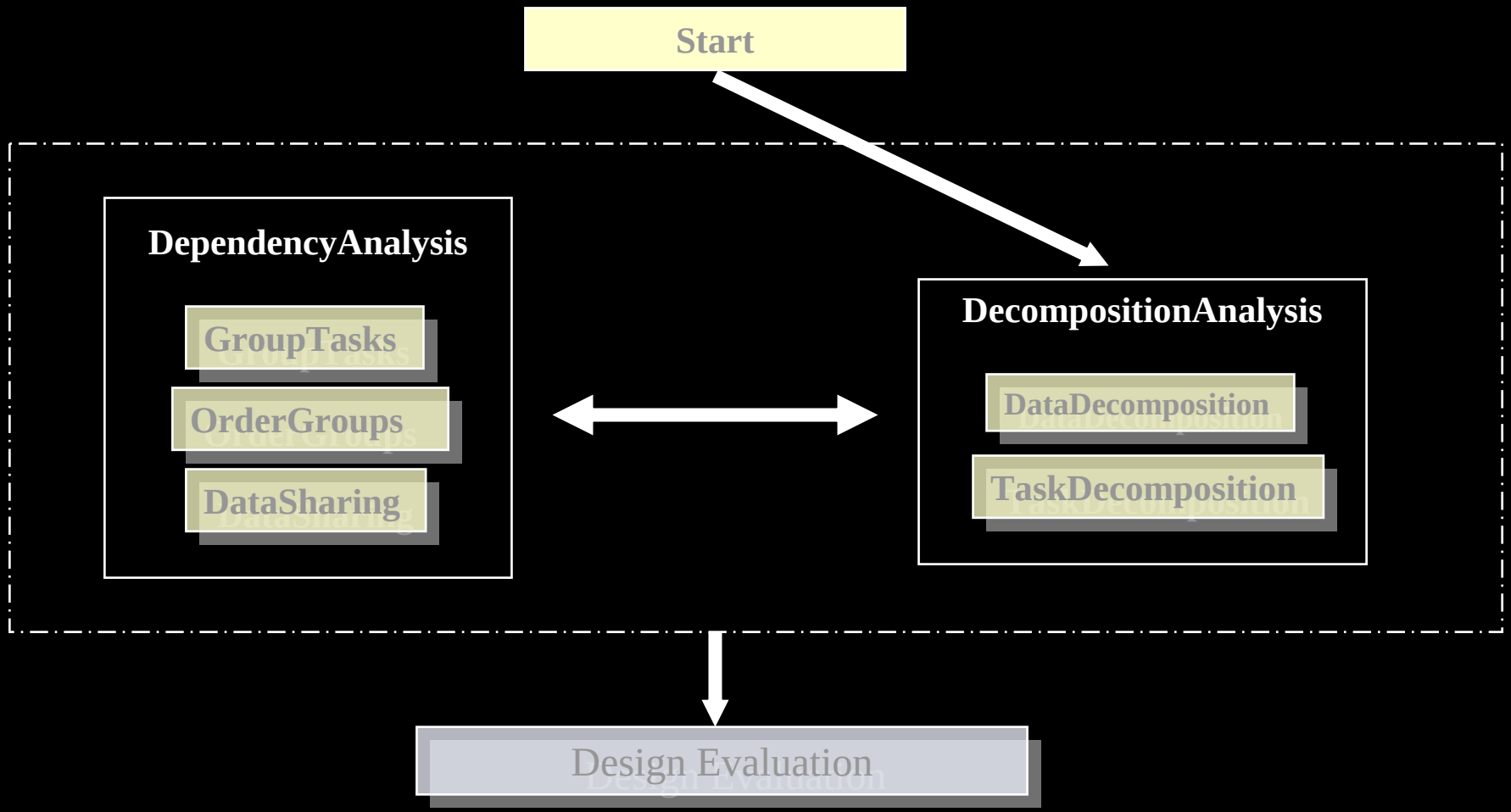
```
Program SPMD_Emb_Par ()
{  Program SPMD_Emb_Par ()
   {  Program SPMD_Emb_Par ()
      {  Program SPMD_Emb_Par ()
         {
            TYPE *tmp, *func();
            global_array Data(TYPE);
            global_array Res(TYPE);
            int Num = get_num_procs();
            int id = get_proc_id();
            if (id==0) setup_problem(N, Data);
            for (int I= ID; I<N;I=I+Num){
                tmp = func(I, Data);
                Res.accumulate( tmp);
            }
         }
      }
   }
}
```

Units of execution + new shared data for extracted dependencies

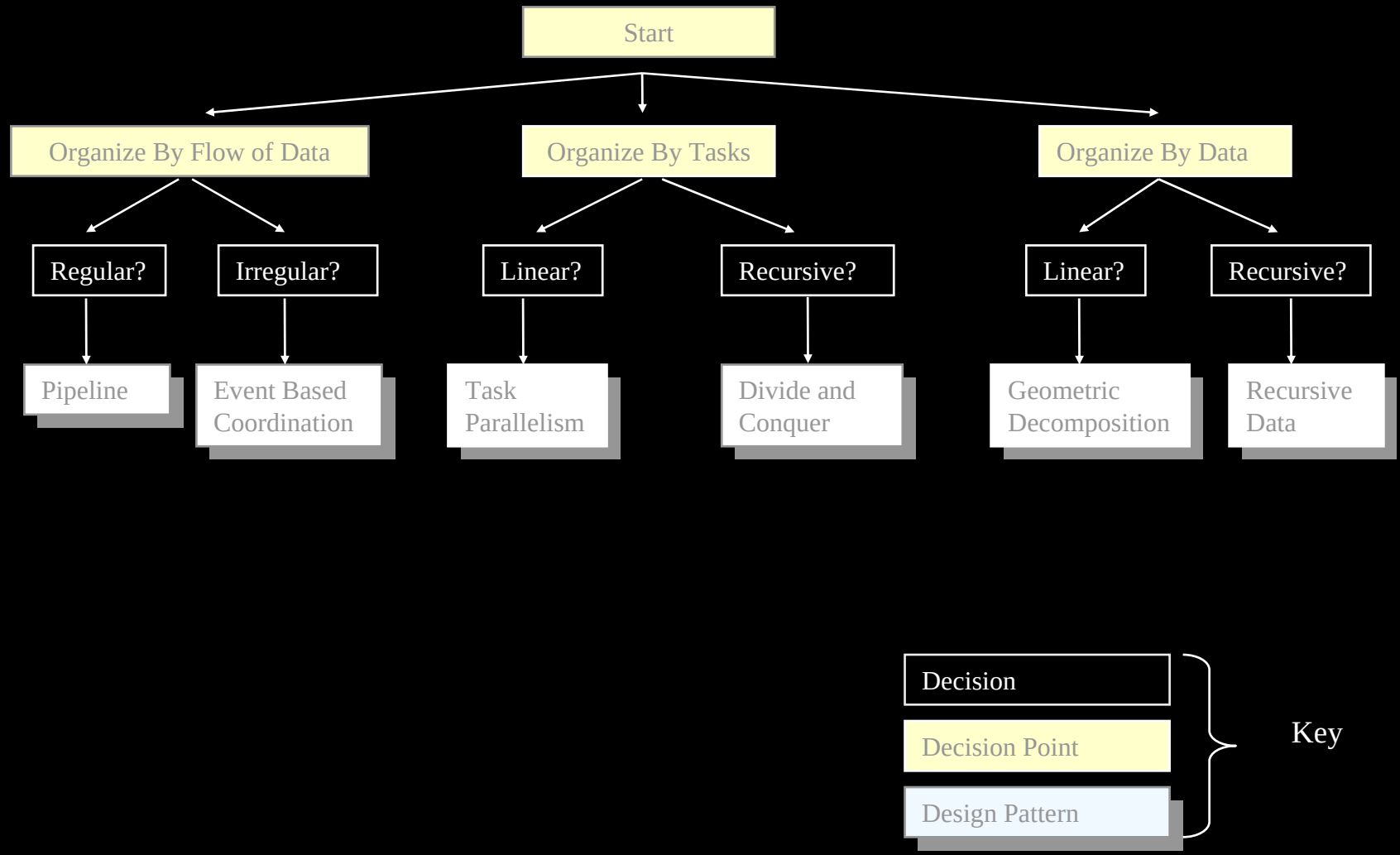
Corresponding source code

The FindingConcurrency Design Space

Start with a specification that solves the original problem -- finish with the problem decomposed into tasks, shared data, and a partial ordering.

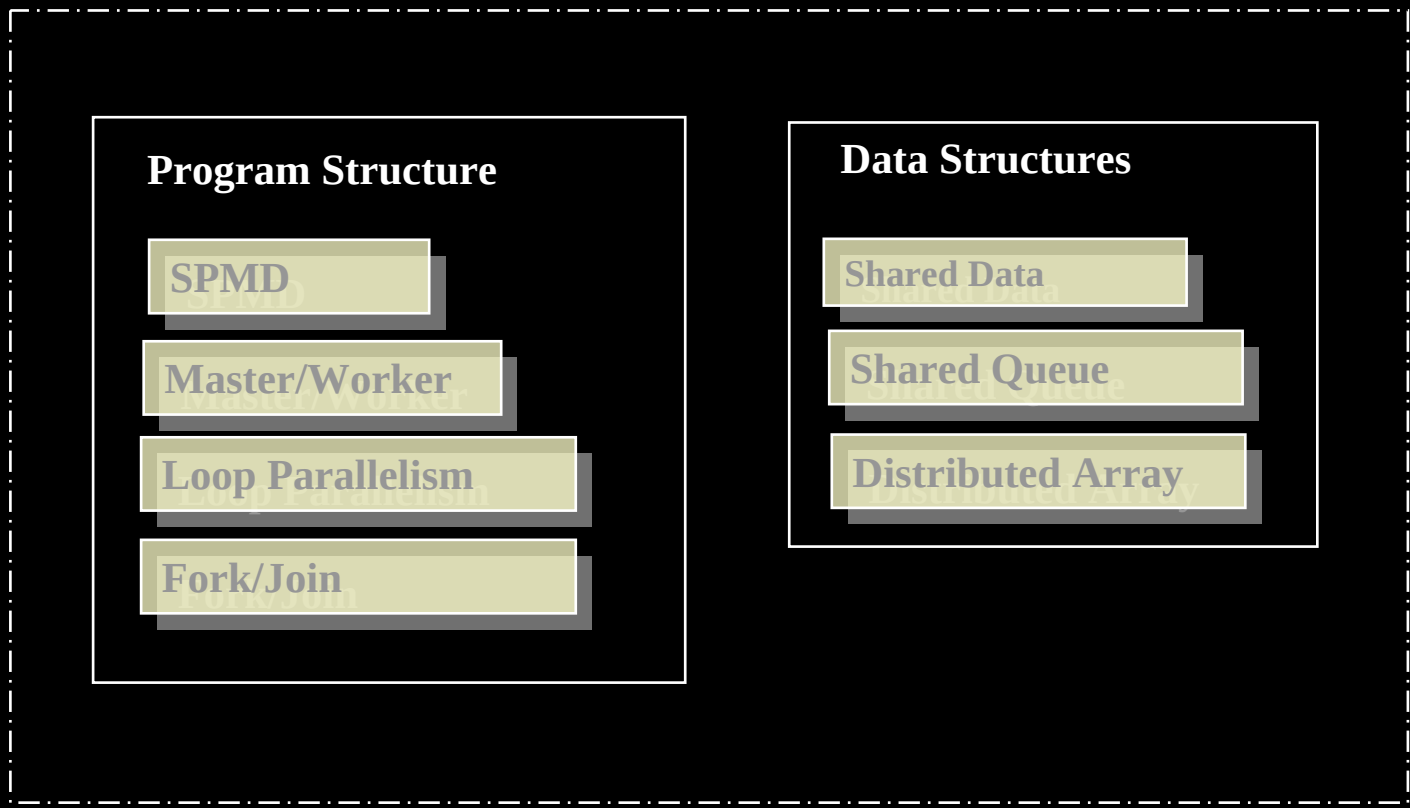


The Algorithm Structure Design Space



The Supporting Structures Design Space

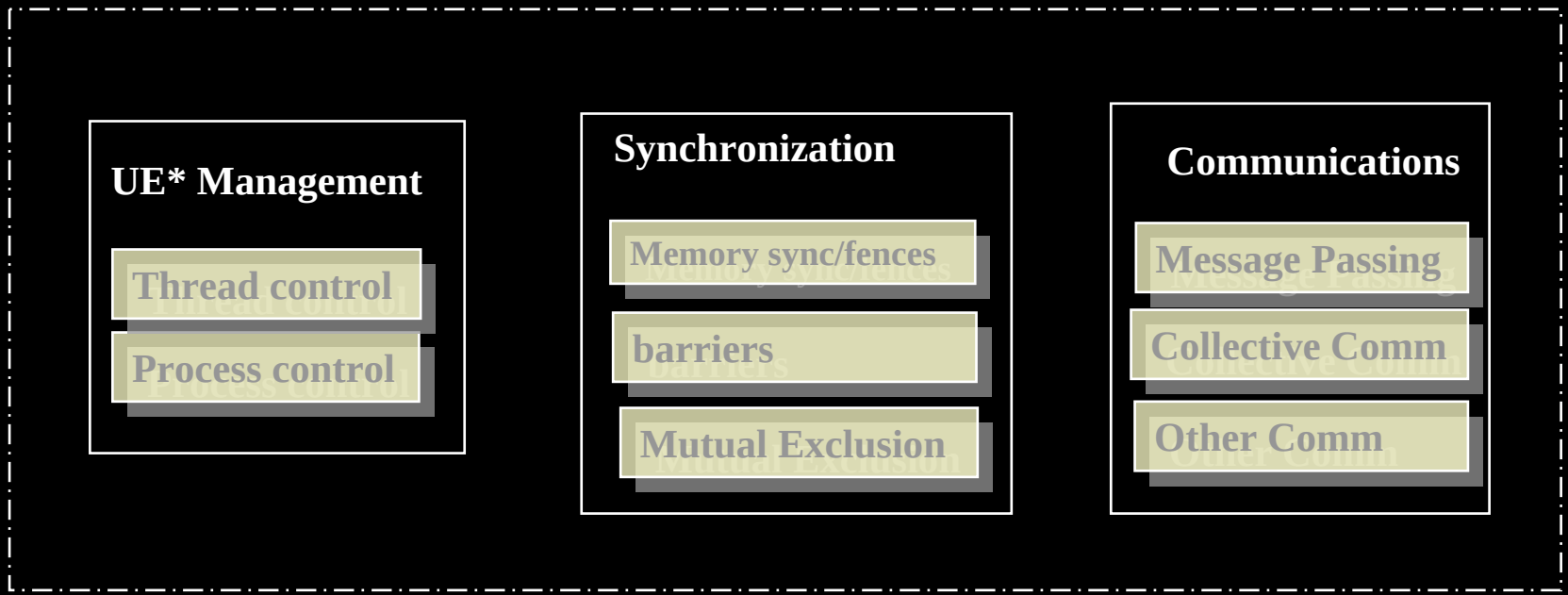
High level constructs impacting large scale organization of the source code.



The Implementation Mechanisms Design Space

Low level constructs implementing specific constructs used in parallel computing. Examples in Java, OpenMP and MPI.

These are not properly design patterns, but they are included to make the pattern language self-contained.



* UE = Unit of execution

The SPMD Pattern

- Problem:
  - How do you structure a parallel program to make interactions between UEs manageable yet easy to integrate with the core computation?
- Forces
  - Fewer programs are easier to manage, but complex algorithms often need very different instruction streams on each UE
  - Balance the conflicting needs of scalability, maintainability, and portability.
- Solution
  - Use a single program for all the UEs.
  - Keep it simple ... use the UE's ID to select different pathways through the program.
  - Keep interactions between UEs explicit and at a minimum.

Why HPC Programming Patterns?

- Document common practice in the design and implementation of high performance programs
  - Education
  - Common language for practitioners
- Help design parallel languages
  - Parallel programming languages should express succinctly prevalent patterns
- Guide design of tools
  - refactoring tools
  - program generators

The Task Coarsening Pattern

- Problem:
  - How do you increase task size so as to avoid overheads.
- Forces
  - Size of task – small size to reduce imbalance vs. large size to reduce scheduling overhead and get surface/volume effect.
- Solution
  - Pick a number of tasks that is equal to number of resources if tasks have equal size or small multiple of number of resources if tasks have unequal size.
  - Remove replications of variables
  - Replace inter-task communication/synchronization with control & data dependencies.
  - Use static scheduling (bin-packing) if tasks have known size or dynamic scheduling if they have unknown size.
- Can compilers/run-time do this automatically?
- Can have refactoring tool to do transformation

What is still missing

- Patterns that deal with performance tuning strategies
  - locality enhancement, load balancing, communication minimization, communication granularity
- Patterns that deal with higher-level problem mapping decisions
  - spatial vs. spectral, domain decomposition, dynamic refinement...
- ...

Future

- Patterns are a valuable way to encode expertise:
  - Can be used to accelerate the transition from novice to expert.
  - Documents insights into how people create parallel algorithms … thereby helping guide design of new parallel programming environments.
- But to be truly valuable, we need a community consensus.
  - The parallel patterns book is a starting point, but there is a huge amount of work left to do.
  - Maybe we need our own patterns workshop:
    - Par PLoP™.

Workshop on Patterns in HPC

- Workshop held at UIUC (Champaign)
  - May 4-6 2005
- Brought together
  - Experts in the Patterns community
    - Ralph Johnson, of the gang-of-Four patterns book
  - Parallel Application developers
  - Designers of Parallel frameworks and libraries
- Unique format
  - Patterns writer's workshop
  - Presentations
  - Working groups

Follow-up

- BOF at SC05
- Planned workshop, probably at Sandia in Fall
- Mailing list: patHPC@cs.uiuc.edu
- Web interface: http://lists.cs.uiuc.edu/mailman/listinfo/pathpc/archive