Good Practices in Parallel and Scientific Software

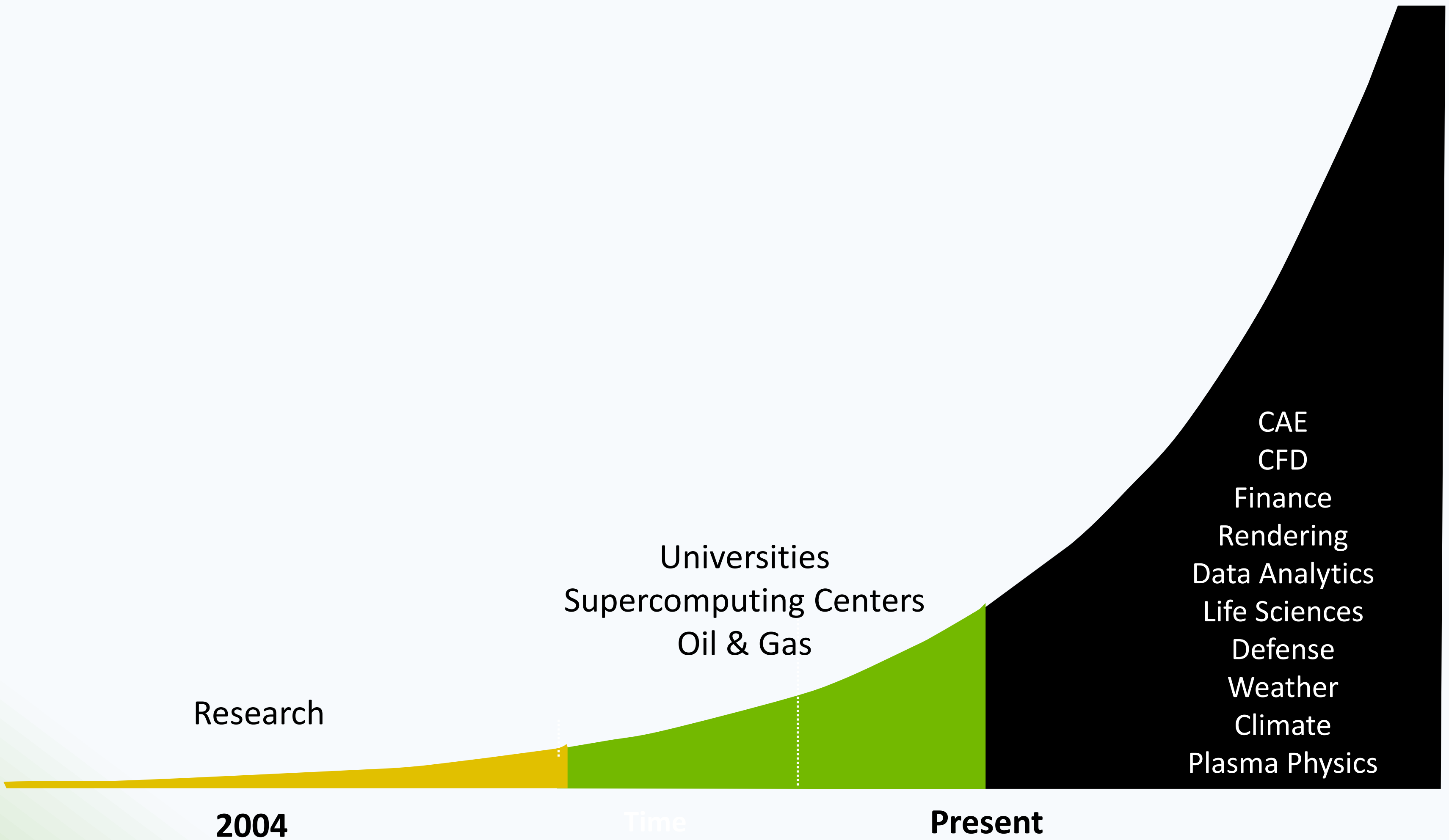
Gabriel Pedraza Ferreira

gpedraza@uis.edu.co

Universidad Industrial de Santander

Super Computación y Cálculo Científico UIS

Parallel Software Development



Some Scientific Software

- Sometimes developed by only one person
- Code has no documentation
- Software has no clear structure
- There is only a copy of code source
- Tools: text editor and a compiler

Software Development Complexity

- Problem Domain
  - Life Science, Finance, Image Process
- Software Architecture
  - Individual Application -> Multiuser
    Reusable Component (3X)
- Technical Complexity
  - Programming Languages, Software
    Software Libraries, Target Platforms

Software Quality

- Reliability
  - Resiliency, Solidity
- Efficiency
  - Performance, Scalability
- Security
  - Vulnerability
- Maintainability
  - Adaptability, Portability, Transferability (one team to other)

Good Practices

- Architecture
- Design
- Coding
- Peer Review
- Testing
- Configuration Management

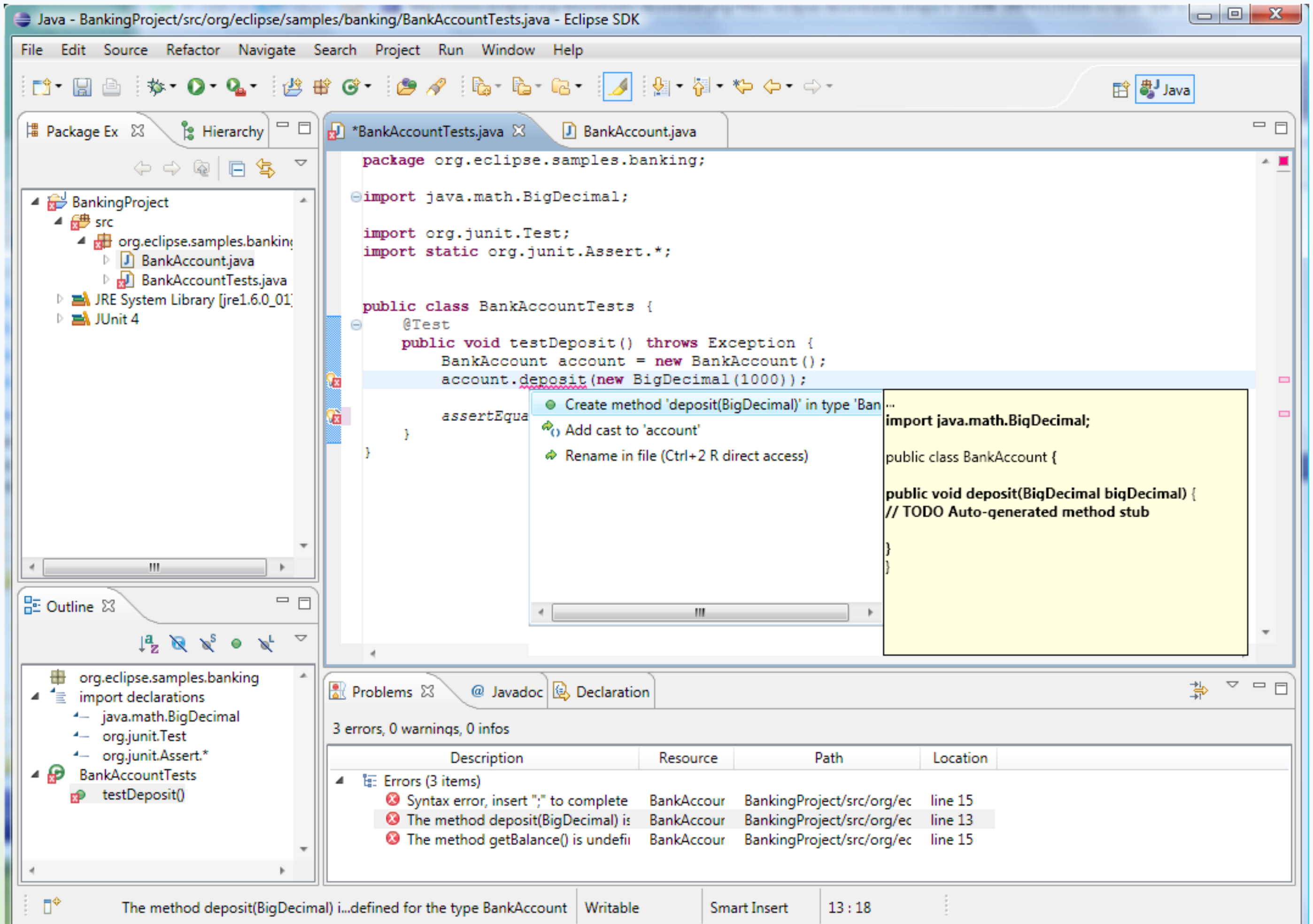
Good Practices
Coding

Good Practices: Coding

- Comment your code
  - Warning DRY
  - i.e. `while (*n++=*i++);`
- Naming conventions
  - Variables a1, a2, a3 … meaning?
- Keep the code simple
  - Another might (will) modify your code in future
- Portability
  - Don’t use hard code (IPs, files, users, urls, ports, etc.)

Good Practices: Coding Tools

- IDE: Integrated Development Environment



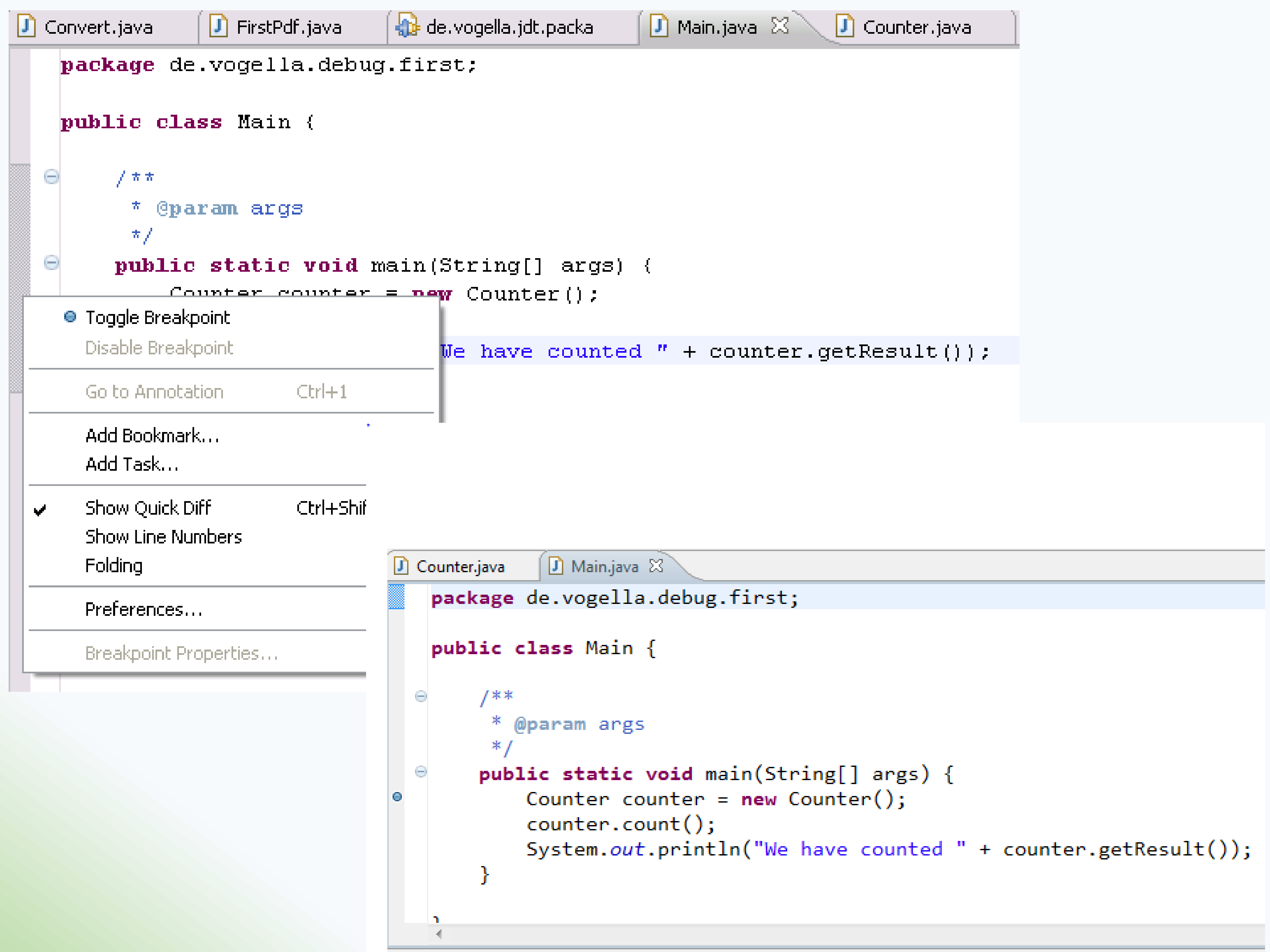
Good Practices
Testing

Good Practices: Testing

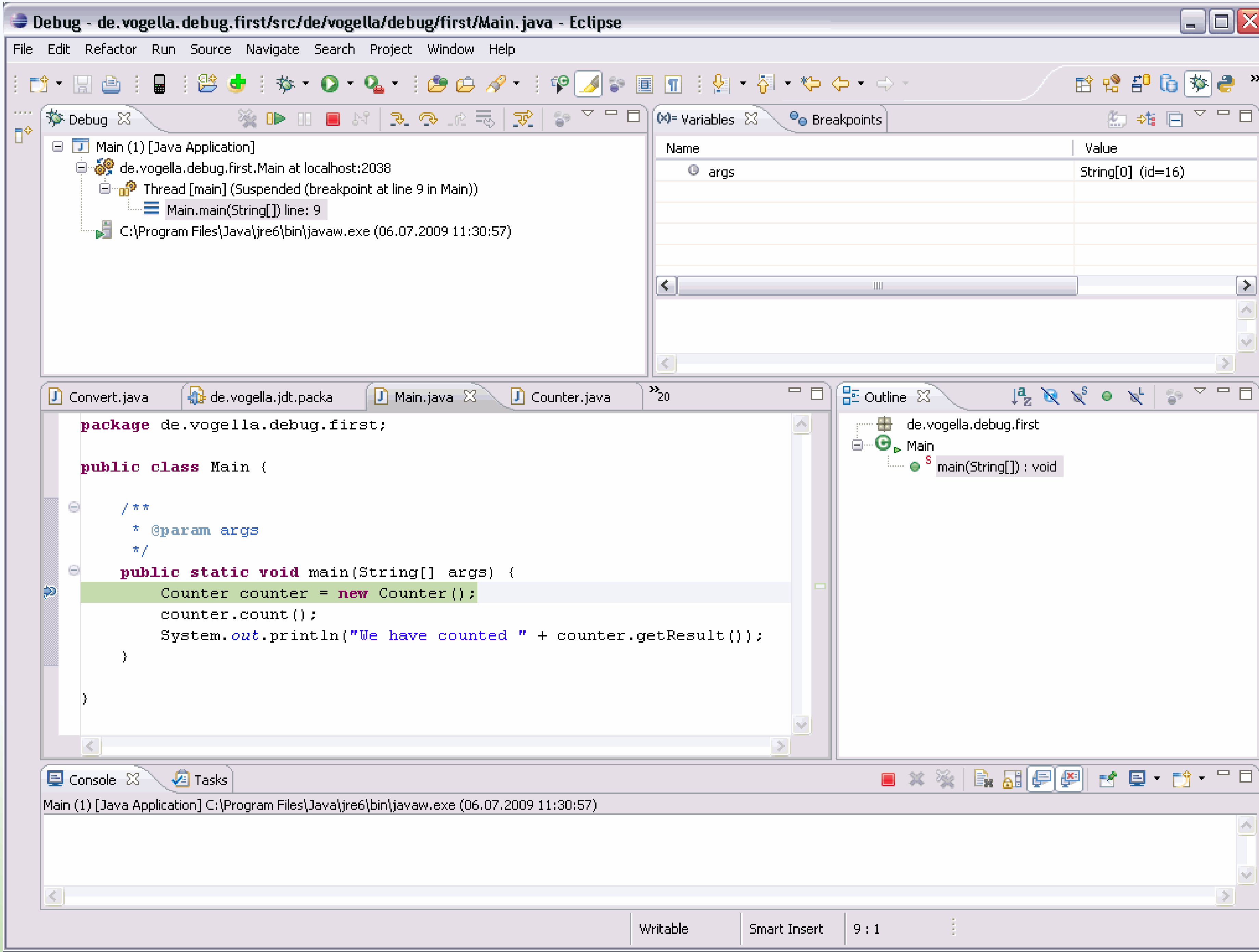
- Testing Methods
  - Static Testing
    - Reviews, Walkthroughs, Inspections
  - Dynamic Testing
    - Execute program with a set of test cases
- Box Approach
  - White Box vs Black Box
- Testing Level
  - Unit, Integration, System

Good Practices: Testing Tools

- Compilers
  - gcc, g++, javac, gfortran, intel compilers…
- Static Analysis Tools
  - Lint, Coccinelle, Pylint…
  - Can be utile to find Heisenbug
- Debuggers
  - GDB is ok but use a front-end



Convert.java
FirstPdf.java
de.vogella.jdt.packa
Main.java
Counter.java
package de.vogella.debug.first;
public class Main {
/**
* @param args
*/
public static void main(String[] args) {
We have counted " + counter.getResult());
Toggle Breakpoint
Disable Breakpoint
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
Show Quick Diff Ctrl+Shif
Show Line Numbers
Folding
Preferences...
Breakpoint Properties...
Counter.java
Main.java
package de.vogella.debug.first;
public class Main {
/**
* @param args
*/
public static void main(String[] args) {
Counter counter = new Counter();
counter.count();
System.out.println("We have counted " + counter.getResult());
}


Debug - de.vogella.debug.first/src/de/vogella/debug/first/Main.java - Eclipse
File Edit Refactor Run Source Navigate Search Project Window Help
Debug
Main (1) [Java Application]
de.vogella.debug.first.Main at localhost:2038
Thread [main] (Suspended (breakpoint at line 9 in Main))
Main.main(String[]) line: 9
C:\Program Files\Java\jre6\bin\javaw.exe (06.07.2009 11:30:57)
Variables
Breakpoints
Name
Value
args
String[0] (id=16)
Convert.java
de.vogella.jdt.packa
Main.java
Counter.java
»20
package de.vogella.debug.first;
public class Main {
/**
* @param args
*/
public static void main(String[] args) {
Counter counter = new Counter();
counter.count();
System.out.println("We have counted " + counter.getResult());
}
}
Outline
de.vogella.debug.first
Main
main(String[]) : void
Console
Tasks
Main (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (06.07.2009 11:30:57)
Writable
Smart Insert
9 : 1

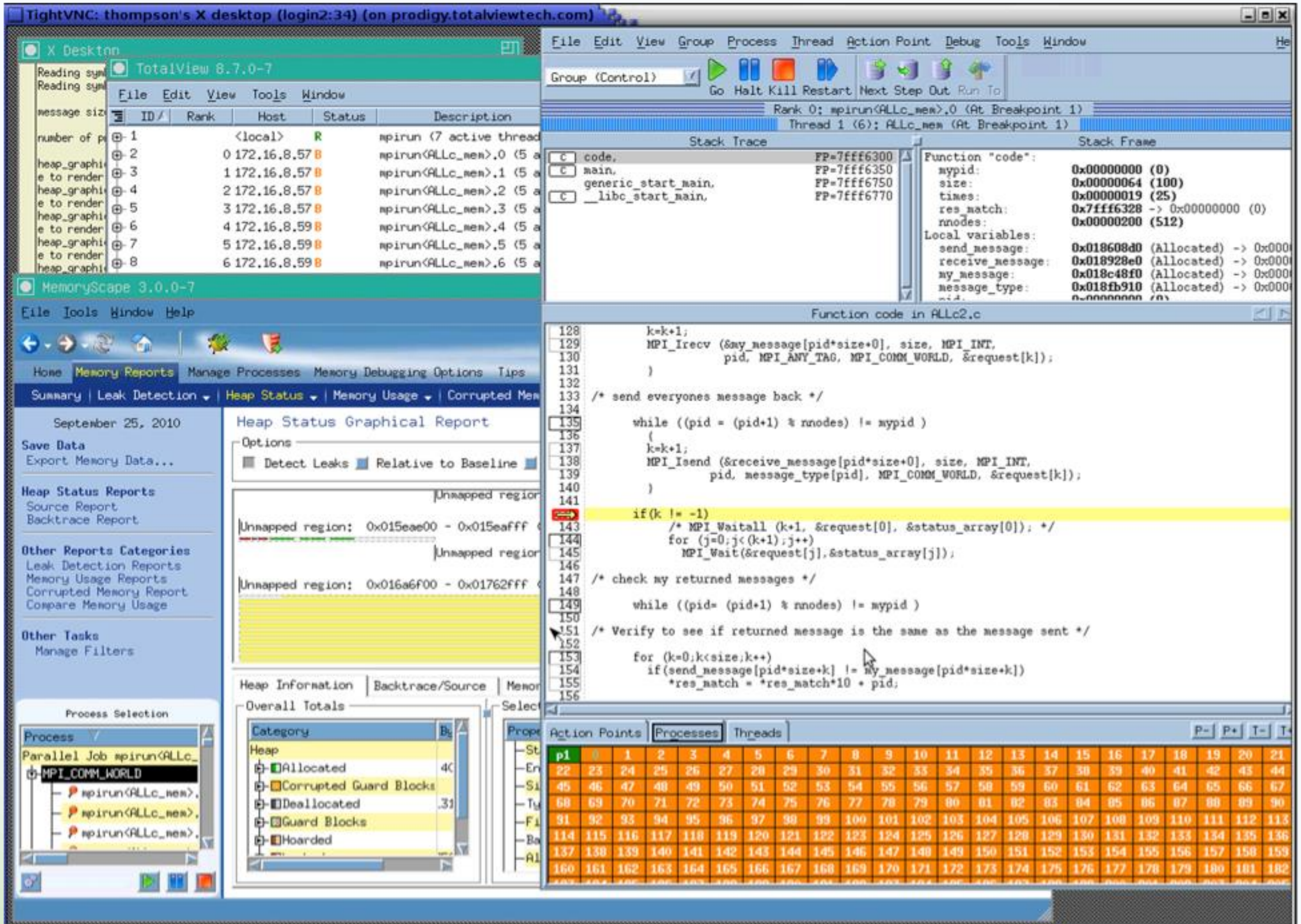
GP: Debugging Parallel Programs

- Parallel programs deals with the usual bugs
- In addition there are timing and synchronization errors
- Parallel bugs often disappear when you add code to try to identify the bug

GP: Visual Debugging Parallel Programs

- A global view of the multiprocessor architecture
  - Processors and communication links
- See which communication links are used
  - Perhaps even change the data in transmission
- Utilization of each processor
  - Can identify blocked processors, deadlock
- “step” through functionality?
  - Lack of a global clock
- Likely won’t help with data races

GP: Debugging Tools – Total View



Erroneous use of Language Features

- Examples
  - Inconsistent parameter types for get/send and put/receive
  - Required function calls
  - Inappropriate choice of functions
- Symptoms
  - Compile-type error (easy to fix)
  - Some defects may surface only under specific conditions: Number of processors, value of input, alignment issues
- Cause
  - Lack of experience with the syntax and semantics of new language features
- Prevention
  - Check unfamiliar language features carefully

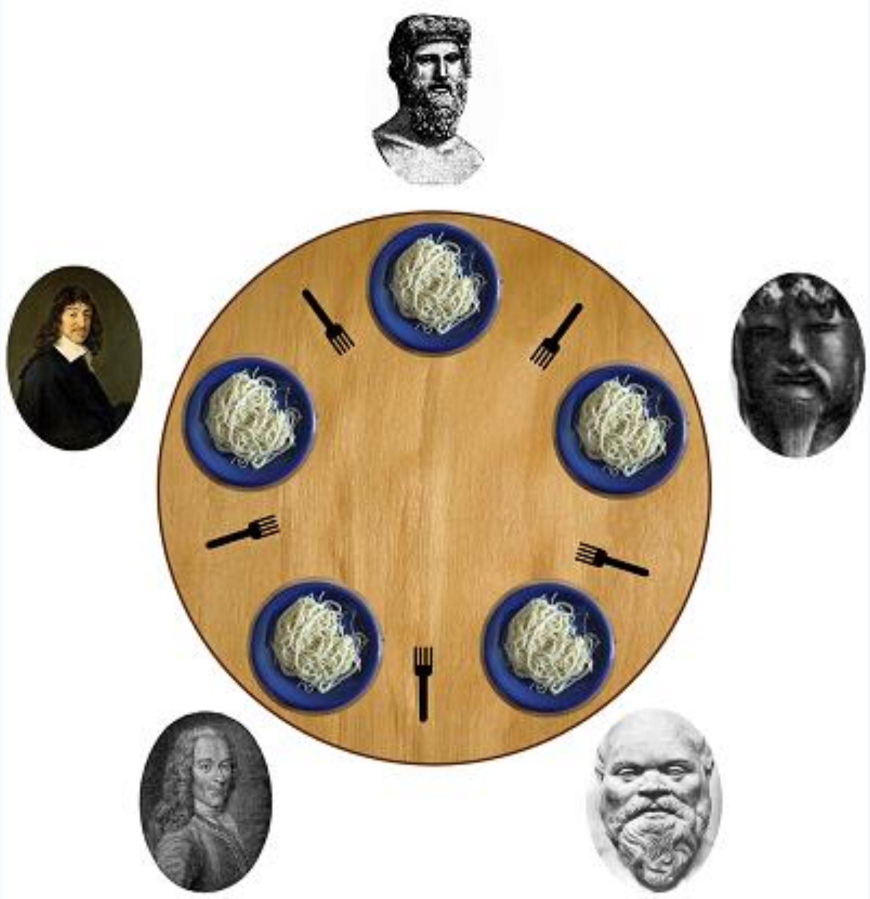
Space Decomposition

- Incorrect mapping between the problem space and the program memory space
- Symptoms
  - Segmentation fault (if array index is out of range)
  - Incorrect or slightly incorrect output
- Cause
  - Mapping in parallel version can be different from that in serial version
  - Array origin is different in every processor
  - Additional memory space for communication can complicate the mapping logic
- Prevention
  - Validate memory allocation carefully when parallelizing code

Deadlock: Dining philosophers problem

a **deadlock** is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

- think until the left fork is available; when it is, pick it up;
- think until the right fork is available; when it is, pick it up;
- when both forks are held, eat for a fixed amount of time;
- then, put the right fork down;
- then, put the left fork down;
- repeat from the beginning.



Race condition

- A timing dependent error involving shared state
- It runs fine most of the time, and from time to time, something weird and unexplained appears

Thread 1	Thread 2		Shared State
			0
Read		<-	0
Increase			0
Write		->	1
	Read	<-	1
	Increase		1
	Write	->	2

Thread 1	Thread 2		Shared State
			0
Read		<-	0
	Read	<-	0
Increase			1
	Increase		1
Write		->	1
	Write	->	1

Synchronization

- Improper coordination between processes
  - Well-known defect type in parallel programming
  - Deadlocks, race conditions
- Symptoms
  - Program hangs
  - Incorrect/non-deterministic output
- Causes
  - Some defects can be very subtle
  - Use of asynchronous (non-blocking) communication can lead to more synchronization defects
- Preventions
  - Make sure that all communication is correctly coordinated

Good Practices
Configuration Management

GP: Configuration Management

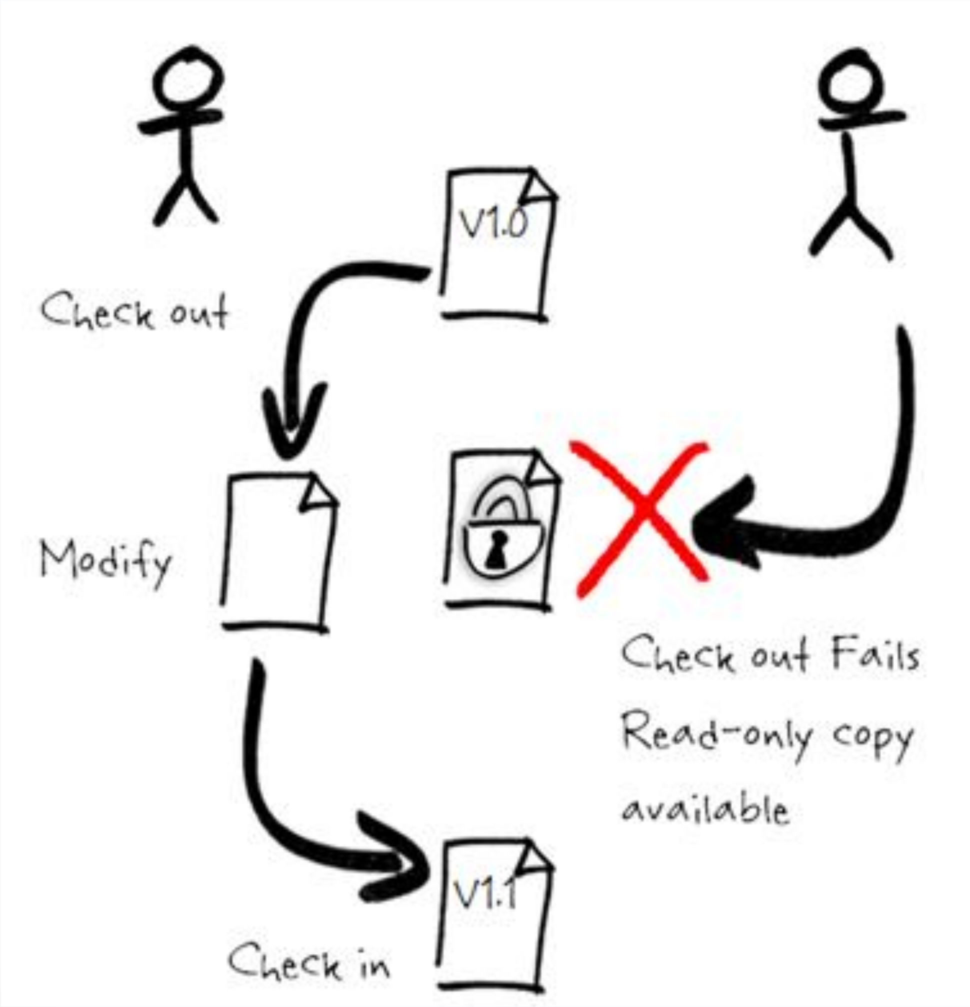
- Track Versions
  - Is that the “last” version?
- Baselines
  - Which version which features?
- Build Management
  - Building a project with 100 source code files
  - Configuration files, several tools
- Bug Tracking
  - Discovery, Assignation, Solution, etc.
- Environment Management
  - Setup of a development and test environments

GP: Track Versions

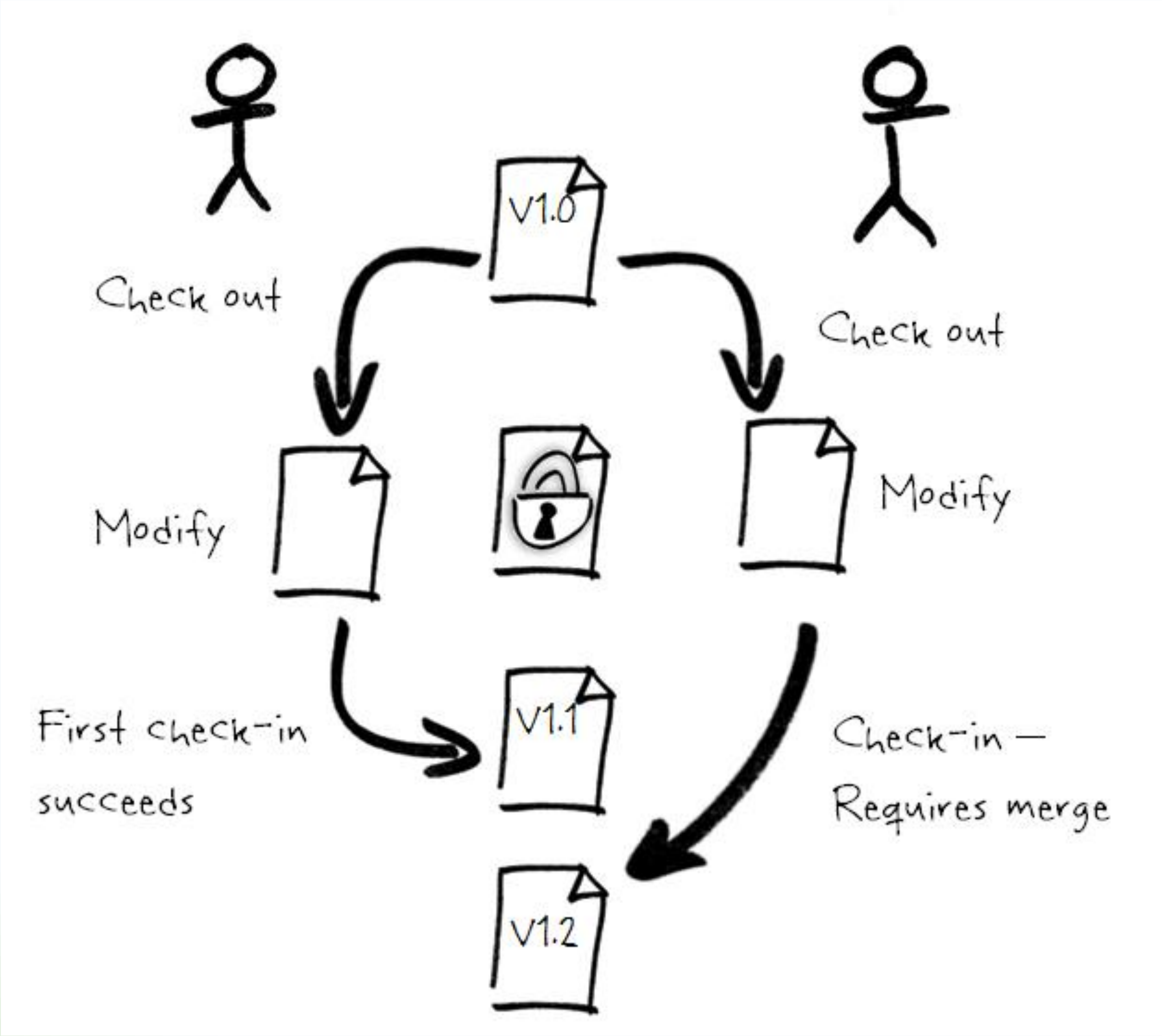
- Manually versioning
  - Pi1.c -> Pi2.c -> Pi2.1.c -> Pi2.1.2.c
  - What is the difference between them?
  - When the changes were made?

- Team development
  - Pi1Juan.c -> Pi1.1Juan.c -> Pi2Juan.c
  - Pi1Claudia.c -> Pi2Claudia.c -> Pi2.1Claudia.c
  - What is the "latest" version?
  - Who made the changes?

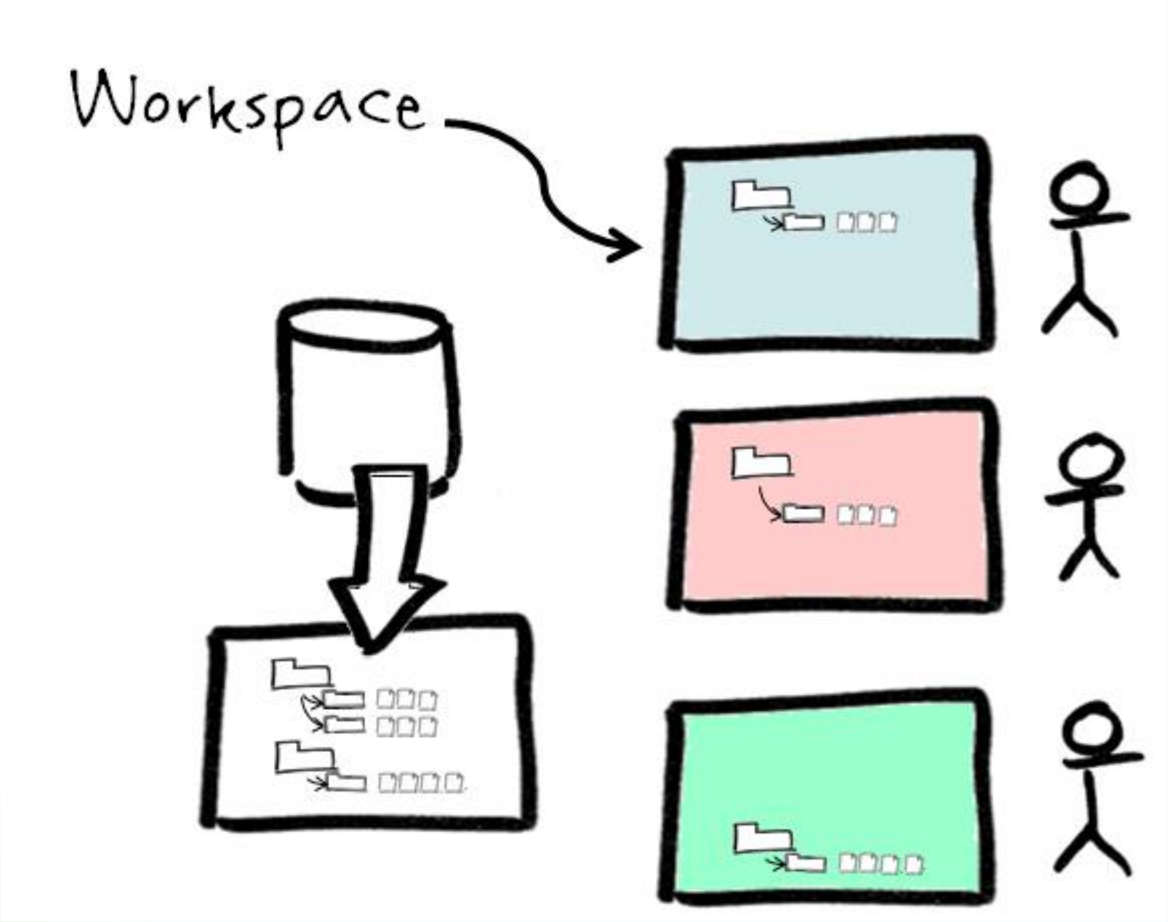
GP: Track Versions



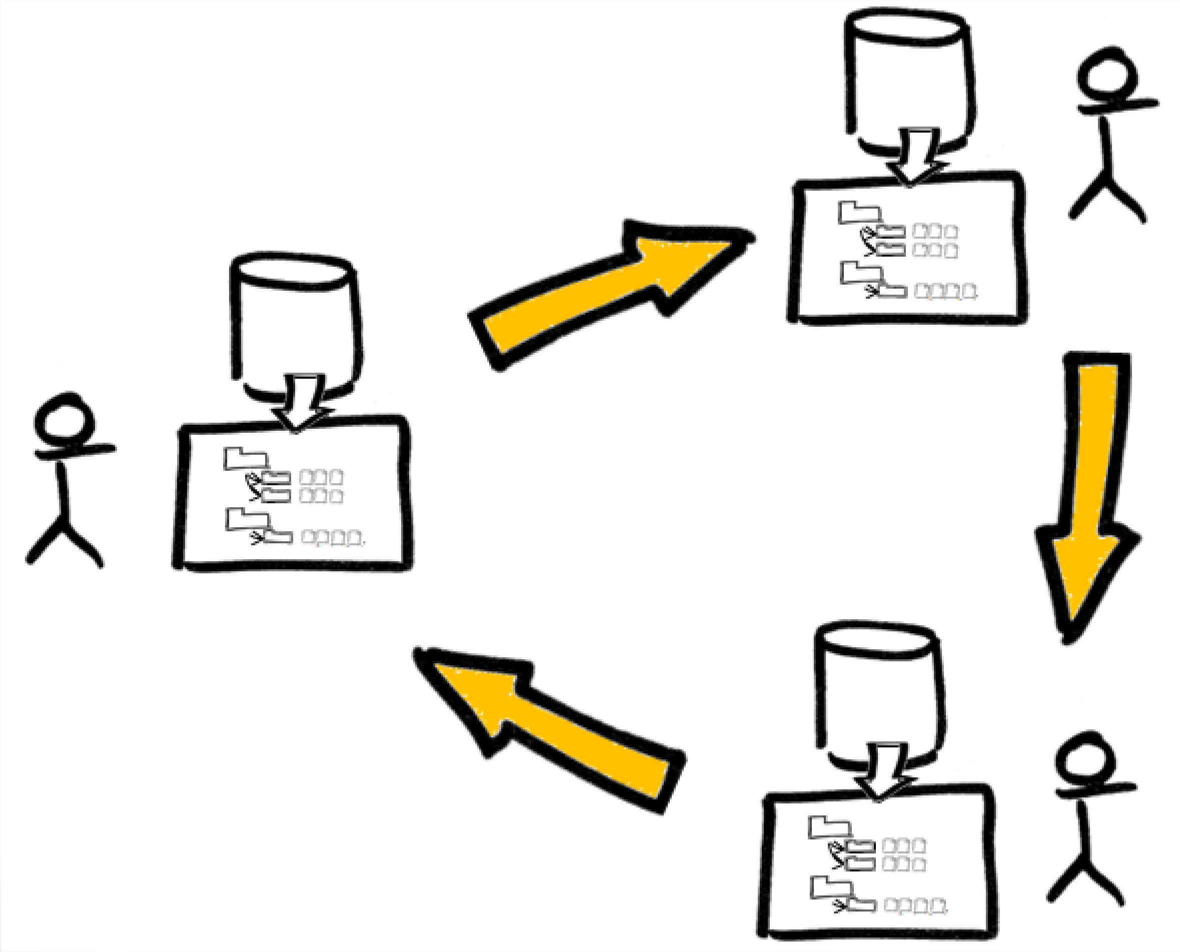
GP: Track Versions



GP: Track Versions



GP: Track Versions



GP: Track Versions Tools

- Version Control

https://github.com/arrayfire

GP: Building Management

- Automation of building process
  - Dependency management
  - Compilation
  - Linkage
  - Documentation Production
  - Artifacts production
  - Code Generation
  - Deployment

GP: Building Management Tools

maven

<APACHE ANT>

Make

http://mrbook.org/blog/tutorials/make/

Good Practices
Architecture

GP: Architecture

- Software Architecture
  - Refers to high level structures of a software systems
  - A good architecture is necessary but not enough
- Reuse is a effective technique in SE
  - Good structuration can be reused
- Structure Patterns
  - Reuse “good” solutions to previous problems

GP: Architecture

- Pattern
  - A pattern is a recurring solution to a standard problem
  - A way of capture and systematize proven practices in any discipline
- Software pattern
  - Function-form relation that occurs in a context, where the function is described in terms of the problem domain terms as a group of unresolved tradeoffs or forces and the form is a structure describe in solution domain terms that achieves a good and acceptable equilibrium among these forces

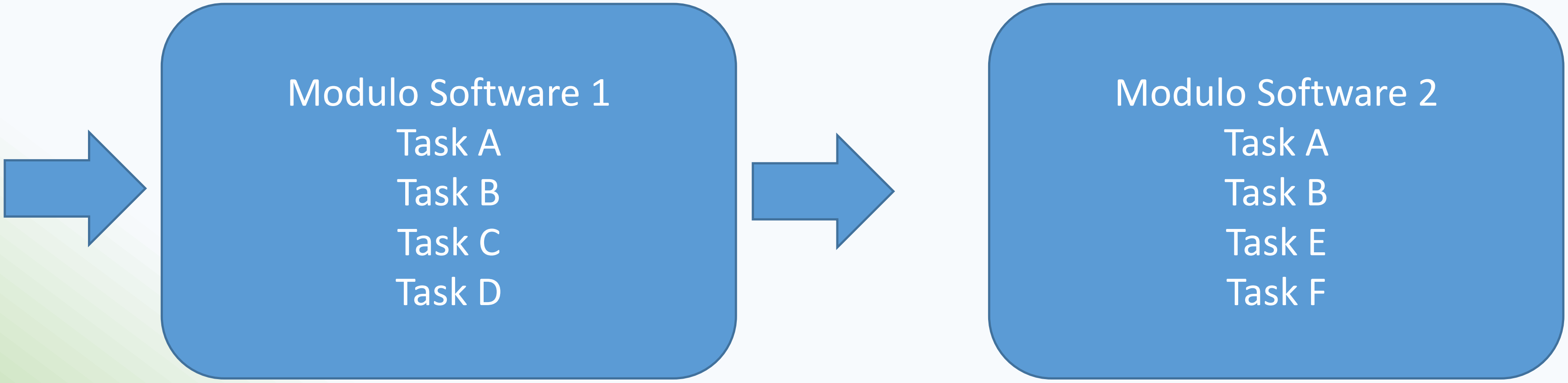
GP: Architecture - Pattern

- Problem
- Context
- Forces
- Solution
- Examples
- Know Uses

GP: Pipe and filter pattern

- Problem

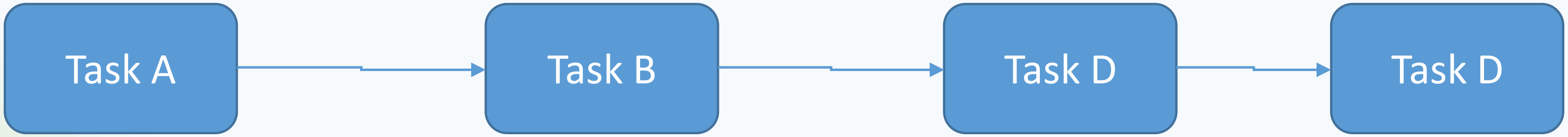
  An algorithm composed of ordered and independent tasks, is required to operate in regular and ordered data. The tasks are ordered but independent of each other, that is, if data is available each task can carried out until completion without interference.



GP: Pipe and filter pattern

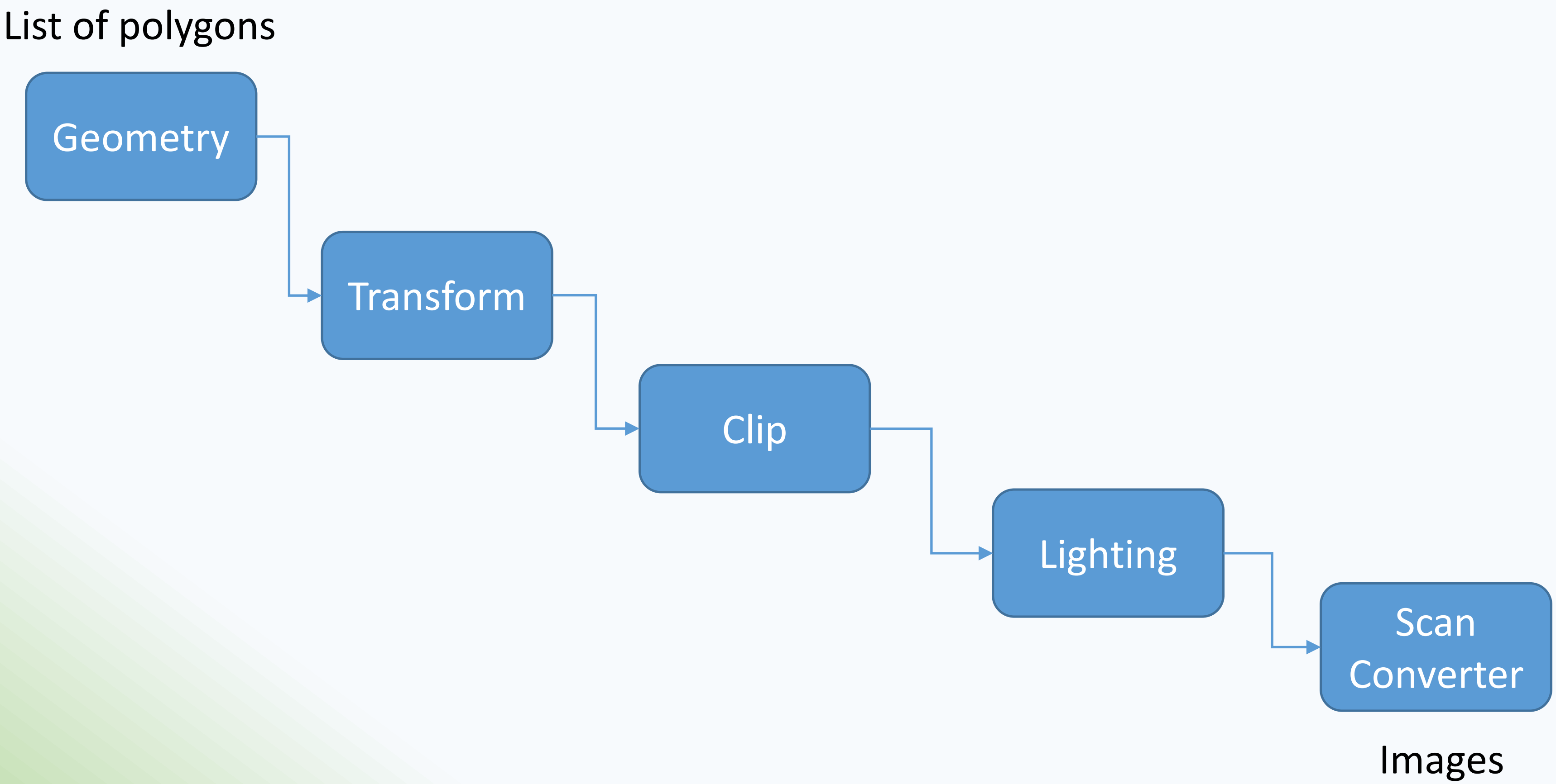
- Solution

  The application should be organized as a series of computation tasks corresponding to the filters, connected by dependencies corresponding to the pipes. The tasks can be seen as vertices in a task graph, and the pipes carrying information from one task to another can be seen as a directed edge in the task graph



GP: Pipe and filter pattern

- Example: Graphics Rendering

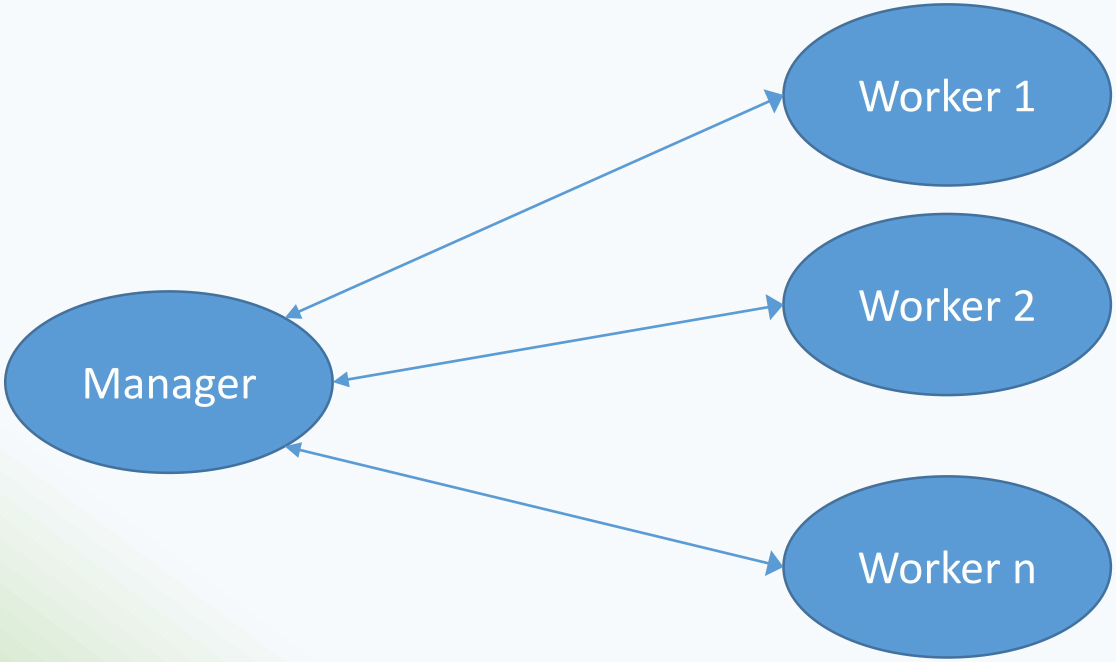


GP: Pipe and filter pattern

- Considering the problem description, granularity and load balancing , the following **forces** should be considered:
  - Preserve the precise order of computations
  - Preserve the data order among of data among all operations
  - Consider the independence among operational steps, whose processing can potentially be carried out on different pieces of data
  - Distribute process evenly among all operational steps
  - Improve performance by decreasing execution time

GP: Manager, Workers Pattern

- Problem
  - The same operation needs to be performed repeatedly on all elements of an ordered dataset. Nevertheless data can be operated on without specific order. It is important, however, to preserve the order of data.



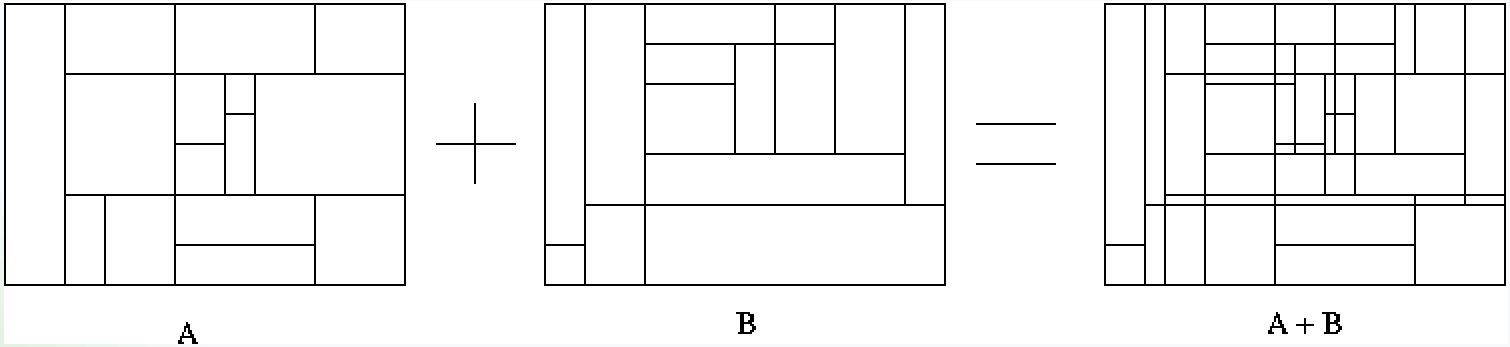
GP: Manager, Workers Pattern

- Solution
  - Introduce activity parallelism by processing multiple datasets at the same time.
  - The solution is structured with a manager and a group of identical workers.
  - The manager is responsible for preserving the order of data.
  - Each worker is capable of performing the same processing on different pieces of data idependently.

GP: Manager, Workers Pattern

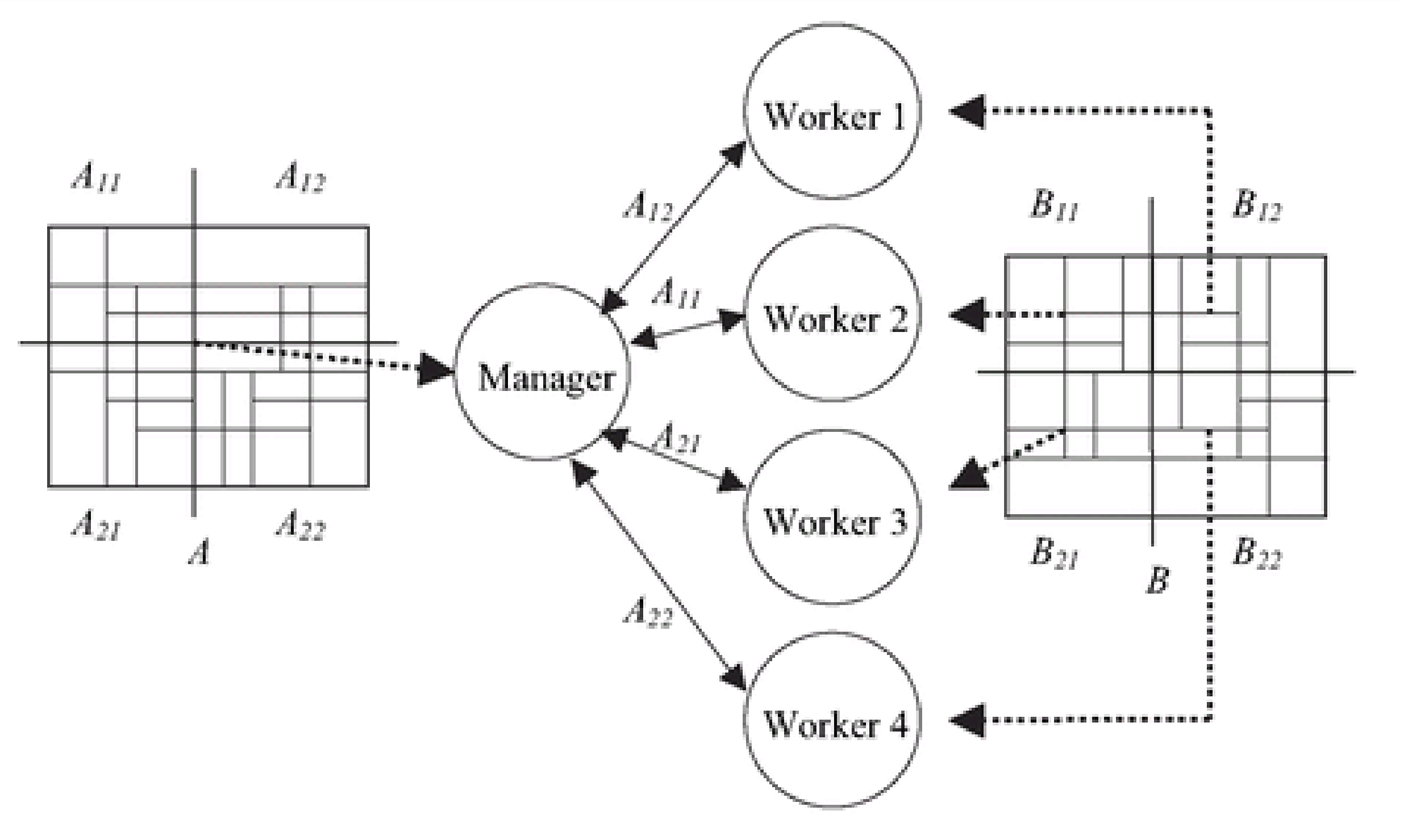
Example: The Polygon overlay problem

- The objective is to obtain the overlay of two rectangular maps A and B



GP: Manager, Workers Pattern

Example: The Polygon overlay problem



GP: Manager, Workers Pattern

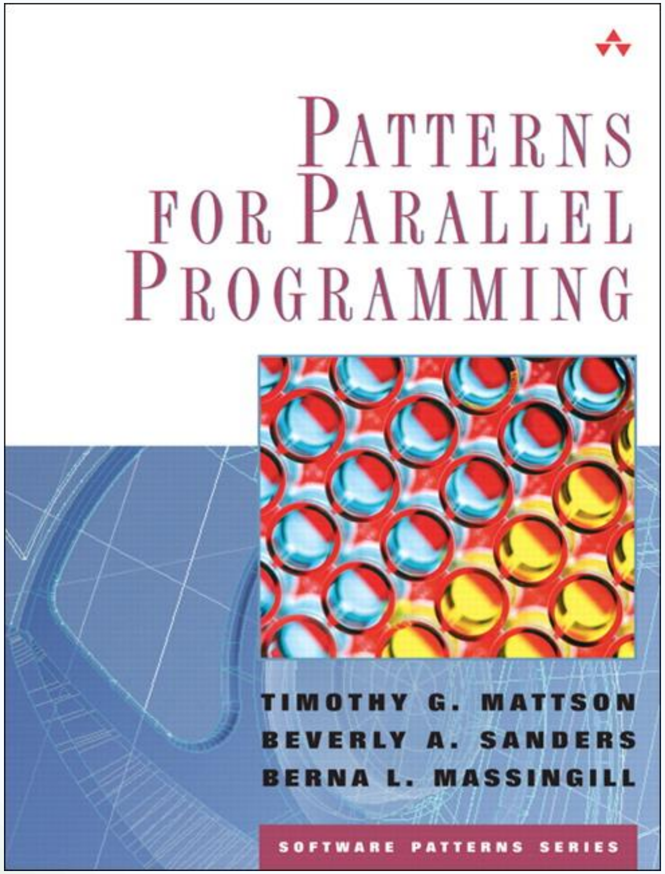
- Forces
  - The order of data must be preserved.
  - The operations must be performed independently on different pieces of data.
  - Data pieces may have different sizes.
  - The solution must scale over the number of processing elements.
  - Mapping the processing elements to processors must take the interconnection among the processors of the hardware platform into account.

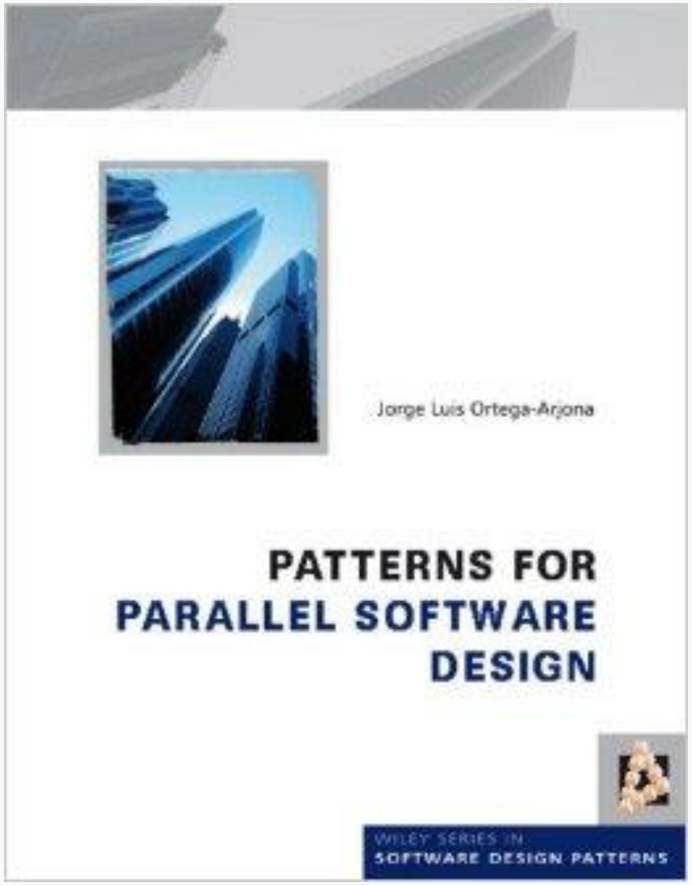
Good Practices Design

Some Patterns to structure algorithms

- SPMD: In an SPMD (Single Program, Multiple Data) program, all UEs execute the same program (Single Program) in parallel, but each has its own set of data (Multiple Data)
- Master Worker: A master process or thread sets up a pool of worker processes or threads and a bag of tasks.
- Loop Parallelism: This pattern addresses the problem of transforming a serial program whose runtime is dominated by a set of compute intensive loops in to a parallel program
- Fork/Join: A main UE forks of f some number of other UEs t hat then continue in parallel to accomplish some portion of the overall work. Often the forking UE waits until the child UEs terminate and join

GP: Design Patterns





Thanks