Universidad de Oviedo

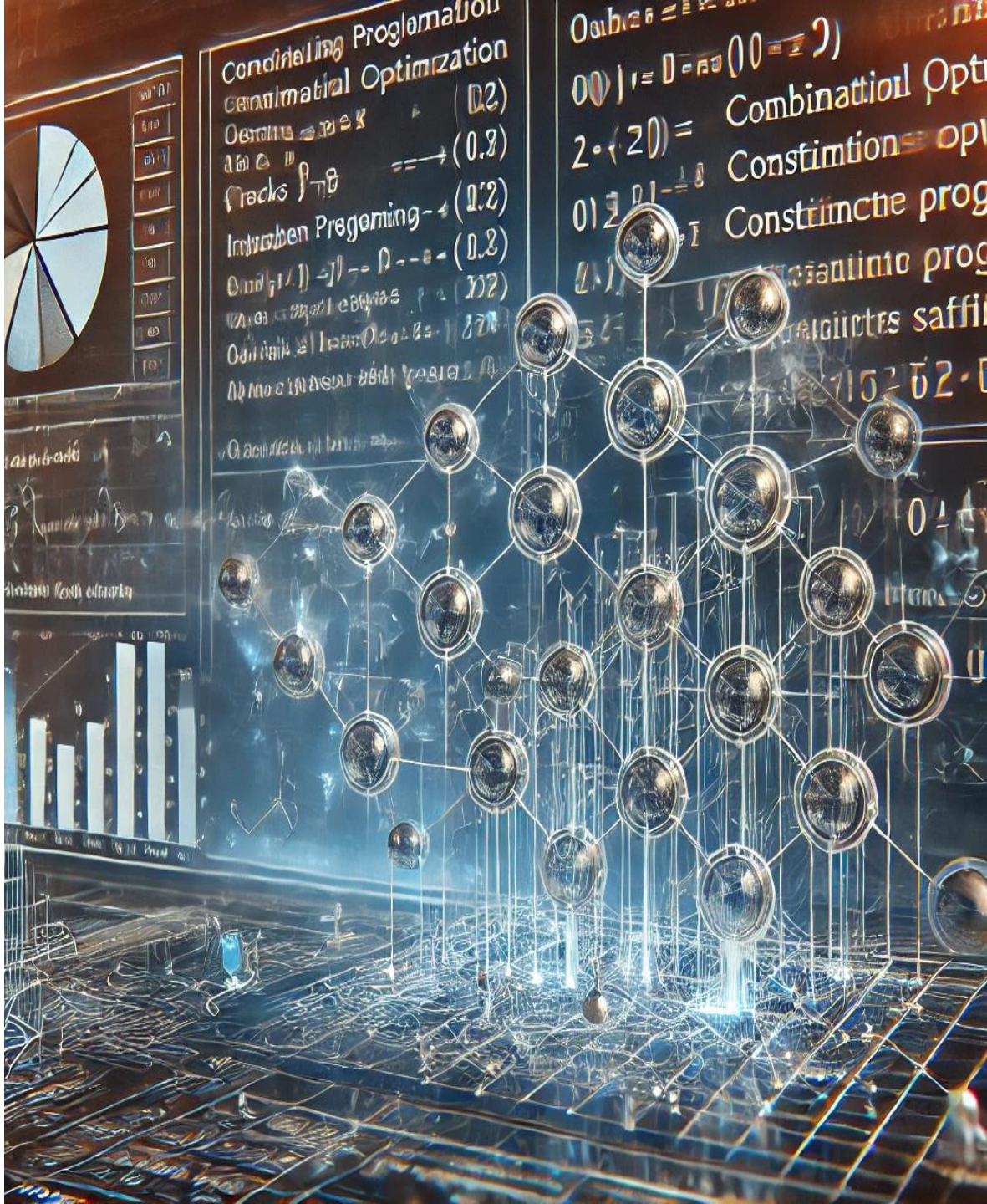
Intelligent Scheduling and Optimization

Técnicas de Inteligencia Artificial para la Optimización y Programación de Recursos

Tema 6: Aplicaciones de Scheduling en la vida real II

Pablo Barredo Gil
uo237136@uniovi.es

Ciencia de la Computación e Inteligencia Artificial
Departamento de Informática



¿Qué vamos a ver hoy?

1. Presentación de un problema real de Scheduling.
2. Conceptos principales del dominio.
3. Identificar las componentes del problema.
4. Identificar los objetivos del problema.
5. Resolución heurística.
6. Resolución con un genético.
7. Operadores.
8. Implementación del genético.

Aplicaciones de Scheduling en la vida real

- Existen múltiples tipos de problemas de optimización.
- Nos centraremos en cómo optimizar la planificación de tareas científicas en el cloud.
- Determinar qué optimizar.
- Qué herramientas usar.
- Analizar los resultados.

¿Qué problema escoger?

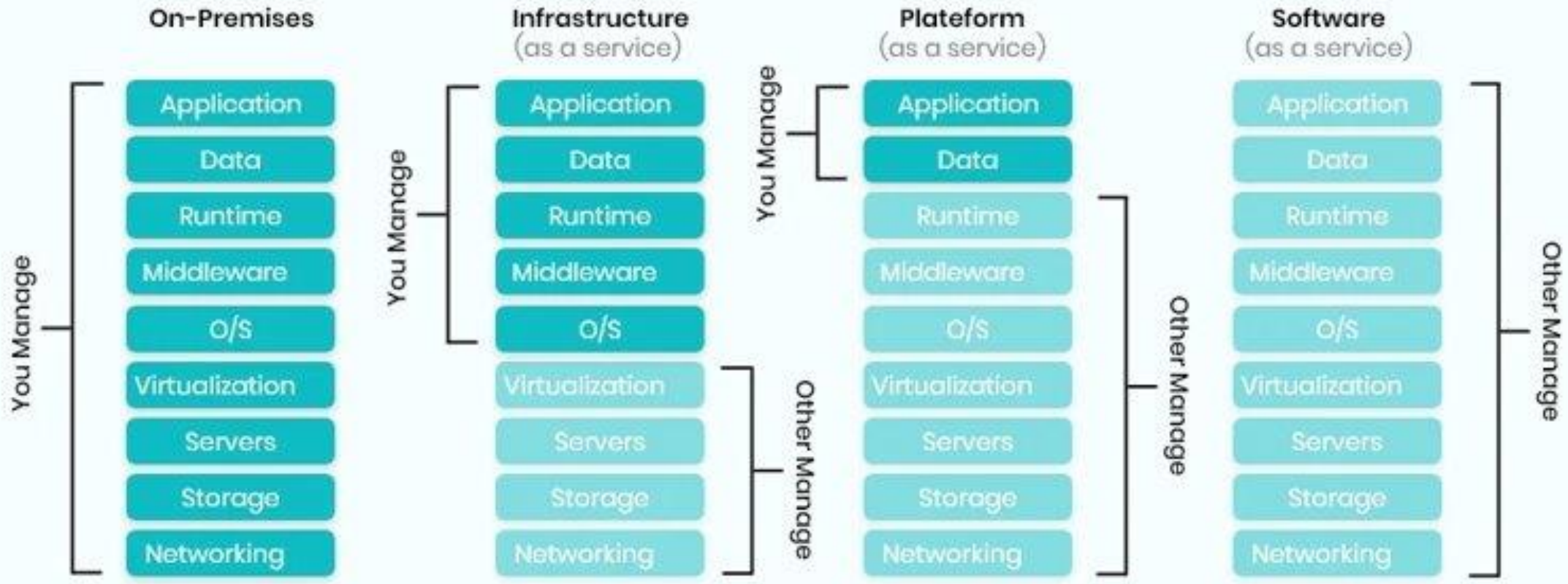
- Identificar una situación no óptima.
- Identificar las métricas a optimizar.
- Identificar los actores y elementos.
- Es necesario tener datos.

- Planificación de Workflows científicos.
- Energía y Makespan (Tiempo de ejecución).
- Servidores y Tareas.
- Repositorio de instancias (WfCommons).

Cloud Computing

- Permite la contratación de cómputo por un periodo de tiempo.
- Existen múltiples servicios y modalidades.
- Destaca por su inmediatez en escala.
- Es posible incurrir en un alto coste.
- Se puede generar una gran dependencia del ecosistema Cloud.

Modelos de Cloud



SaaS (Software as a Service)

PaaS (Platform as a Service)

GitHub

GitLab

IaaS (Infrastructure as a Service)

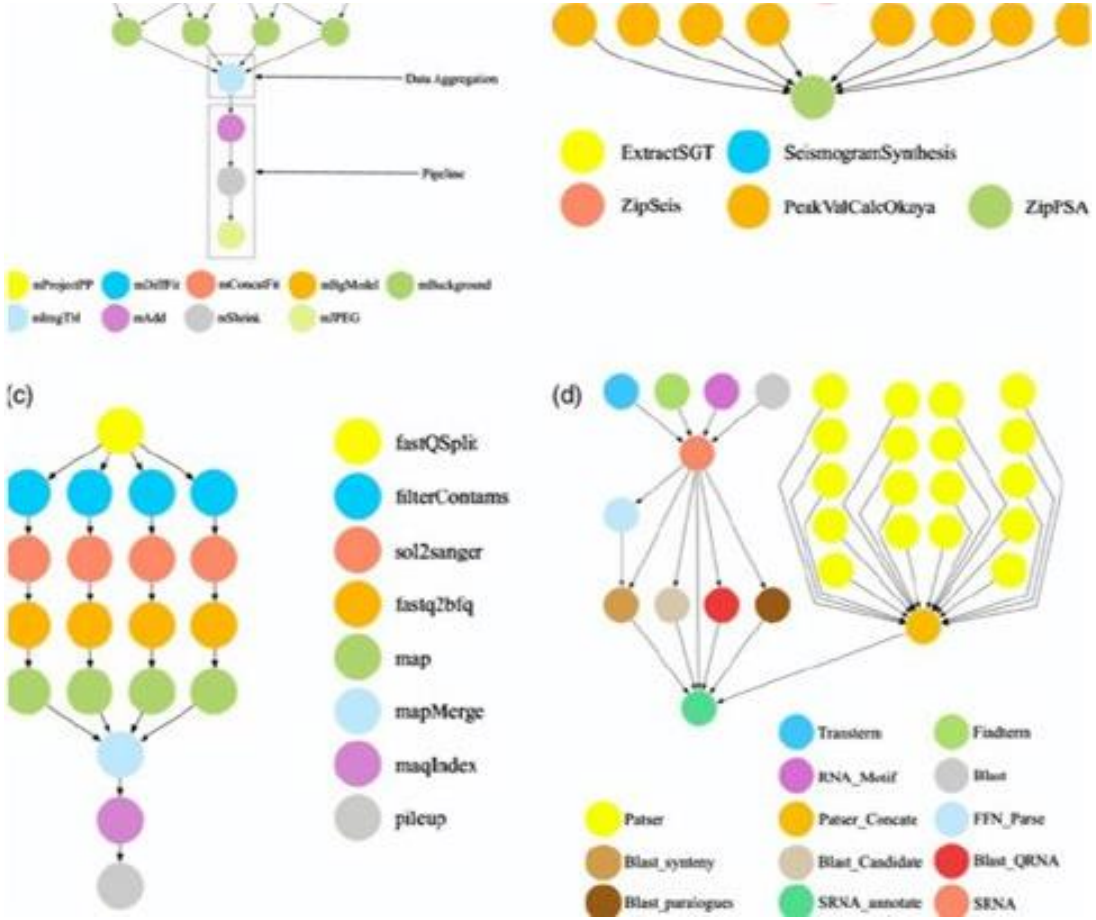
Microsoft Azure

aws

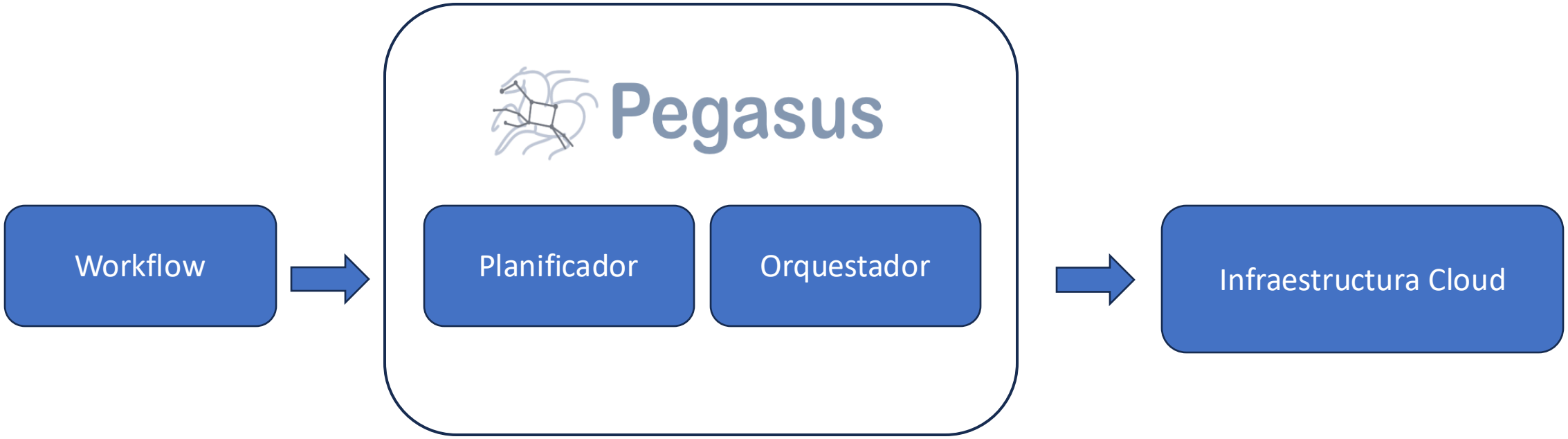
Google Cloud

Describiendo el dominio

- Un workflow es un conjunto de tareas interconectadas.
- Existen dependencias entre tareas.
- Las tareas transmiten información de hijas a padres.
- Hay un gran volumen de tareas.
- Se paralelizan las tareas en diferentes máquinas.



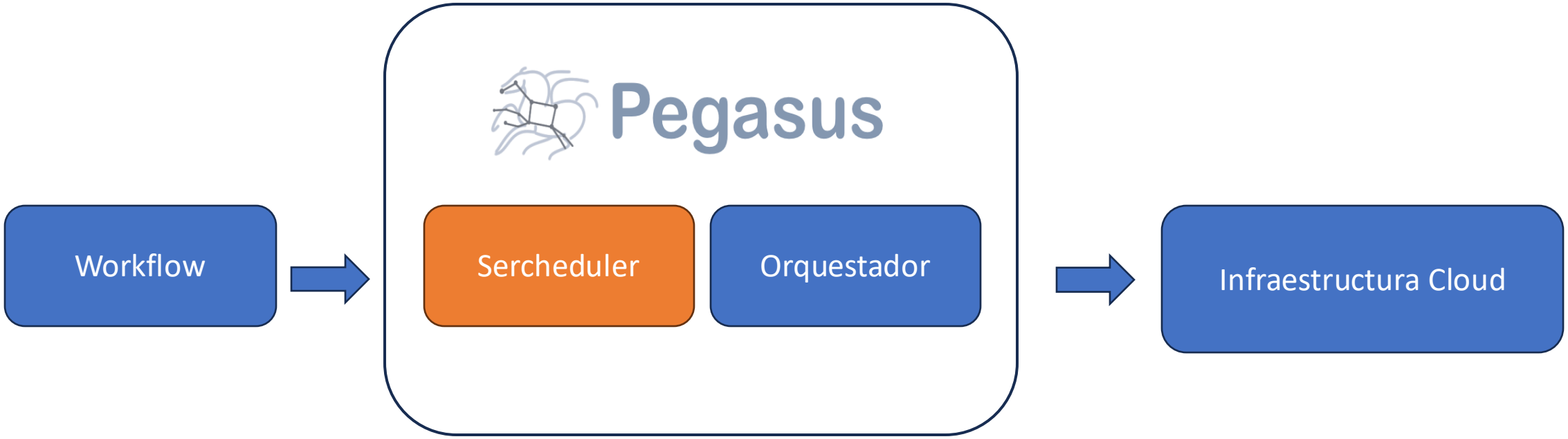
WMS (Workflow Management System)



Problemas con el planificador

- Este tipo de planificadores suelen ser muy simples.
- Round Robin, FIFO, …
- Un buen planificador puede ahorrar mucho tiempo y energ*ía*.

Objetivo de la Tesis

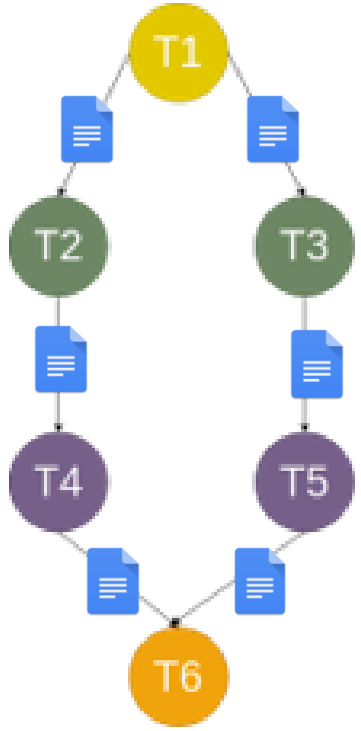


Identificando los actores del problema

- Tenemos que identificar que componentes hay en nuestro problema.
- Que componentes afectan directamente al problema.
- Que propiedades o características tienen esos actores.

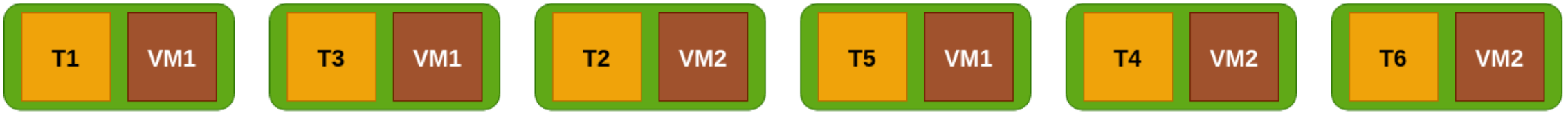
Workflow

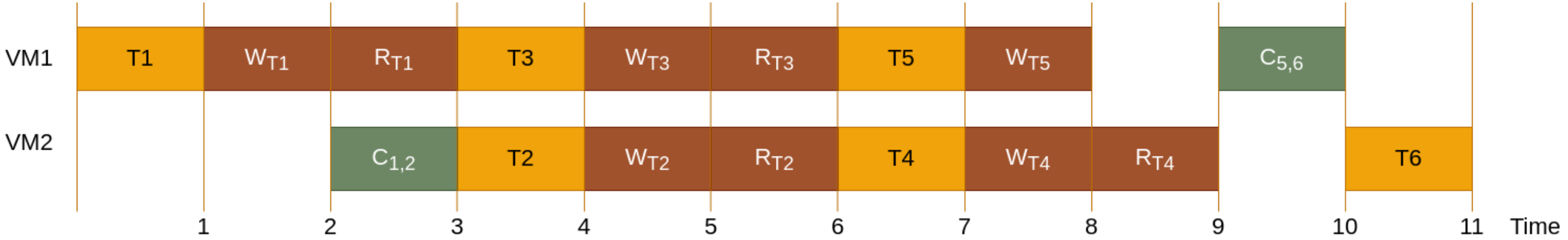
- Es un Grafo Dirigido de tareas:
  - Cada Tarea (nodo) tiene:
    - Un identificador único
    - Carga de computación (GFLOPs)
    - Consumen y generan Información: ficheros (MBs)
  - Dependencias entre tareas (arcos entre nodos)
    - Comunicaciones entre tareas
    - Lectura de ficheros generados por tareas previas



Makespan

- Tiempo en el que se tarda en completar todas las tareas





MKP = 11 TU

Infraestructura: Máquina virtual

- CPU (GFLOPs)
- Velocidad de Disco (MB/sec)
- Velocidad de Red (Mb/sec)
- Energía que consume en standby (W)
- Energía consumida por la CPU y el Disco (W)

Energía

- Cuanta energía consumen todas las máquinas en total.
- Energía activa: Es aquella que se consume cuando se está realizando una tarea, leyendo o escribiendo a disco.
- Energía pasiva: Es la energía consumida simplemente por tener la máquina encendida.

Resolviendo con heurísticos: HEFT

- Heterogeneous Earliest Finish Time.
- Permite resolver el problema de manera rápida.
- Suele ser un resultado aceptable pero nunca óptimo.
- Solo lo aplicamos para minimizar el makespan.

Resolviendo con heurísiticos: HEFT II

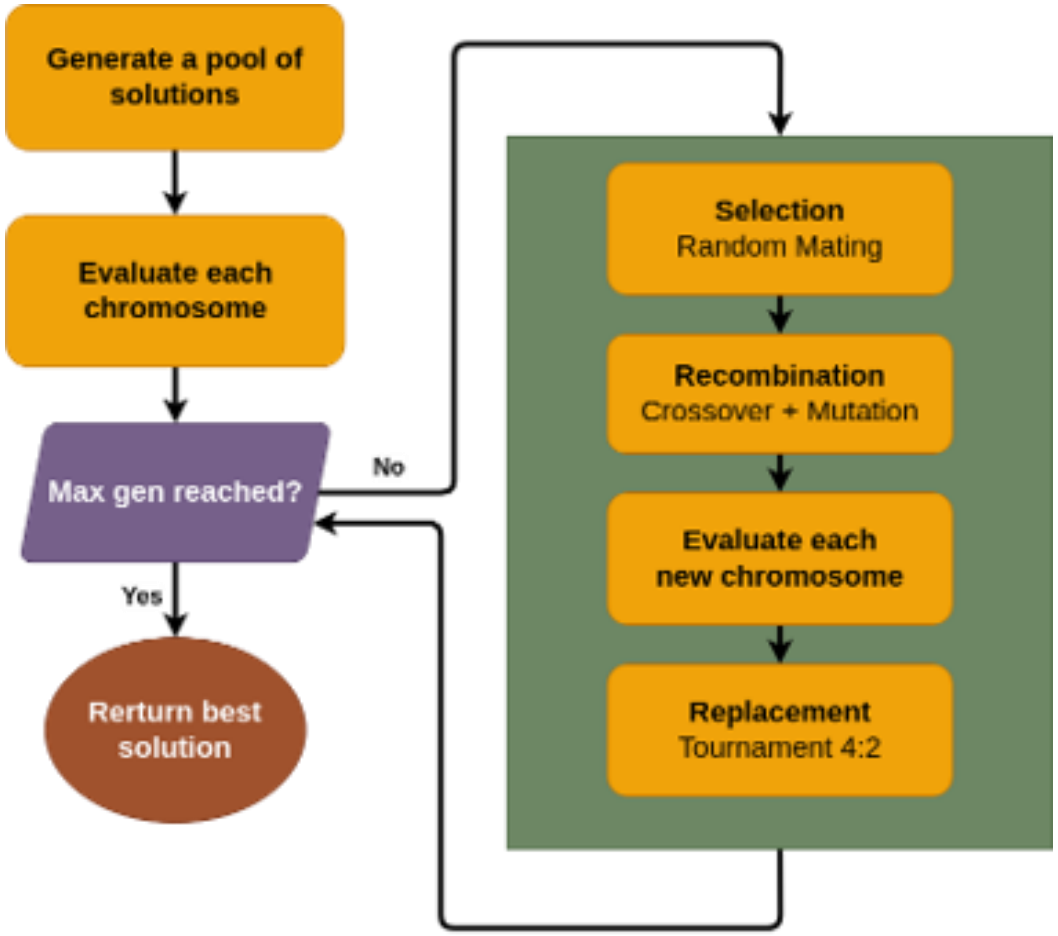
1. Calcular el coste de cada tarea y sus hijos.
2. Ordenar las tareas de mayor a menor prioridad.
3. Iterar por todas las máquinas averiguando el tiempo de ejecución.
4. Asignar la máquina virtual donde termine antes.
5. Construir la planificación.

Algoritmo de inserción

- Al existir dependencias y tener que retrasar la ejecución de las tareas pueden quedar huecos.
- Se debe intentar planificar las tareas en estos huecos siempre que las dependencias de tareas no lo impidan.
- Permite un mejor aprovechamiento de los recursos y disminuye el makespan.

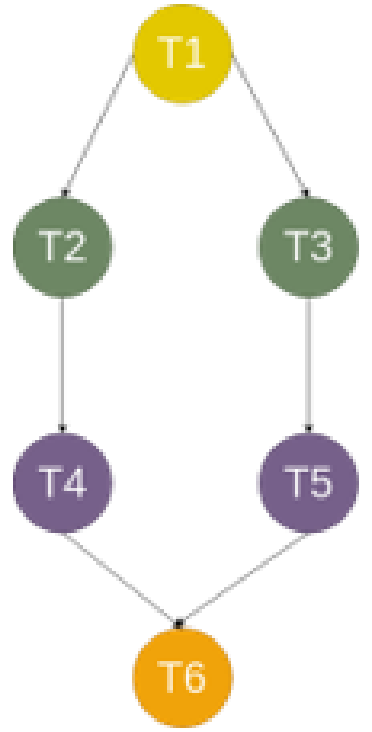
Resolviendo con Algoritmos Genéticos

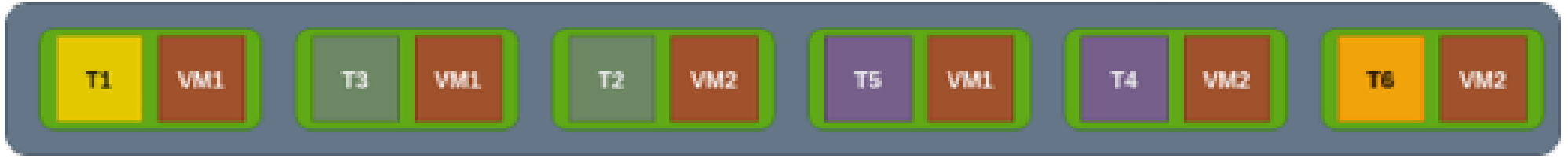
- Ofrece mejores soluciones.
- Requiere un mayor tiempo de ejecución.
- Se puede hacer una implementación mono-objetivo y multi-objetivo de manera sencilla.



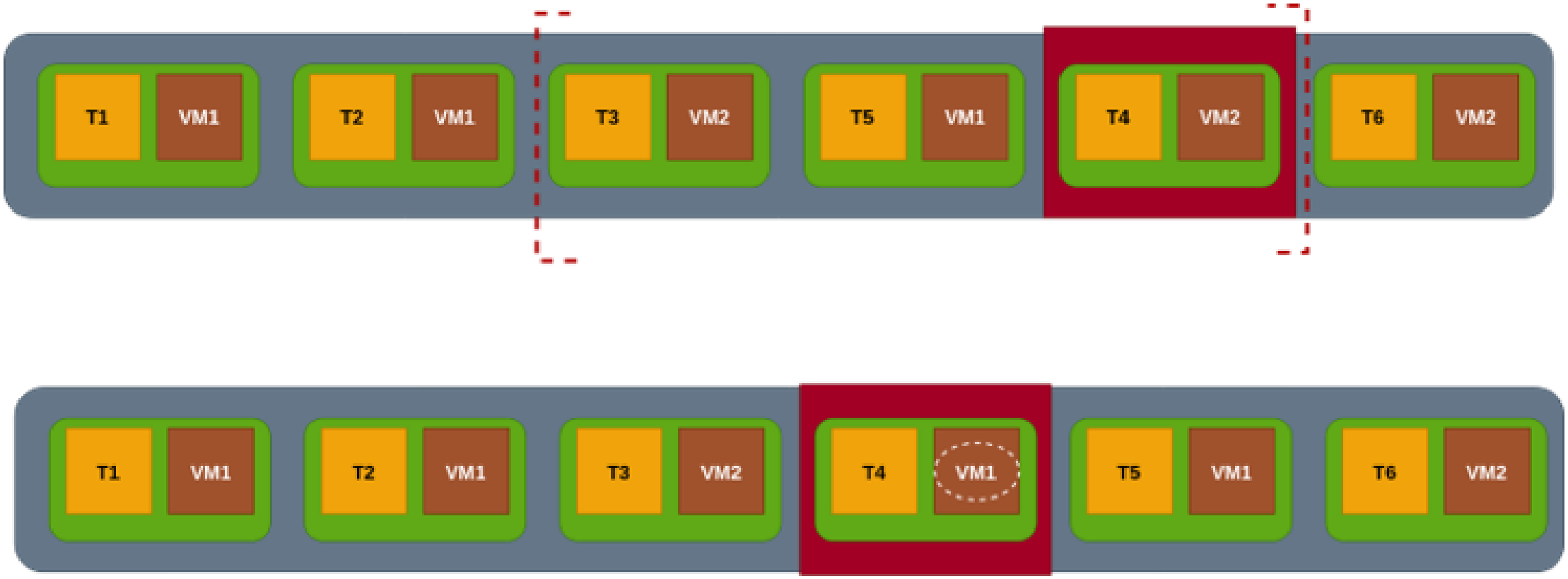
Codificacion de las soluciones

- Establecer el orden de las tareas.
- Asignar una máquina virtual a cada tarea.
- Los hijos no se pueden ejecutar antes que los padres.
- Los padres no se pueden ejecutar después que los hijos.

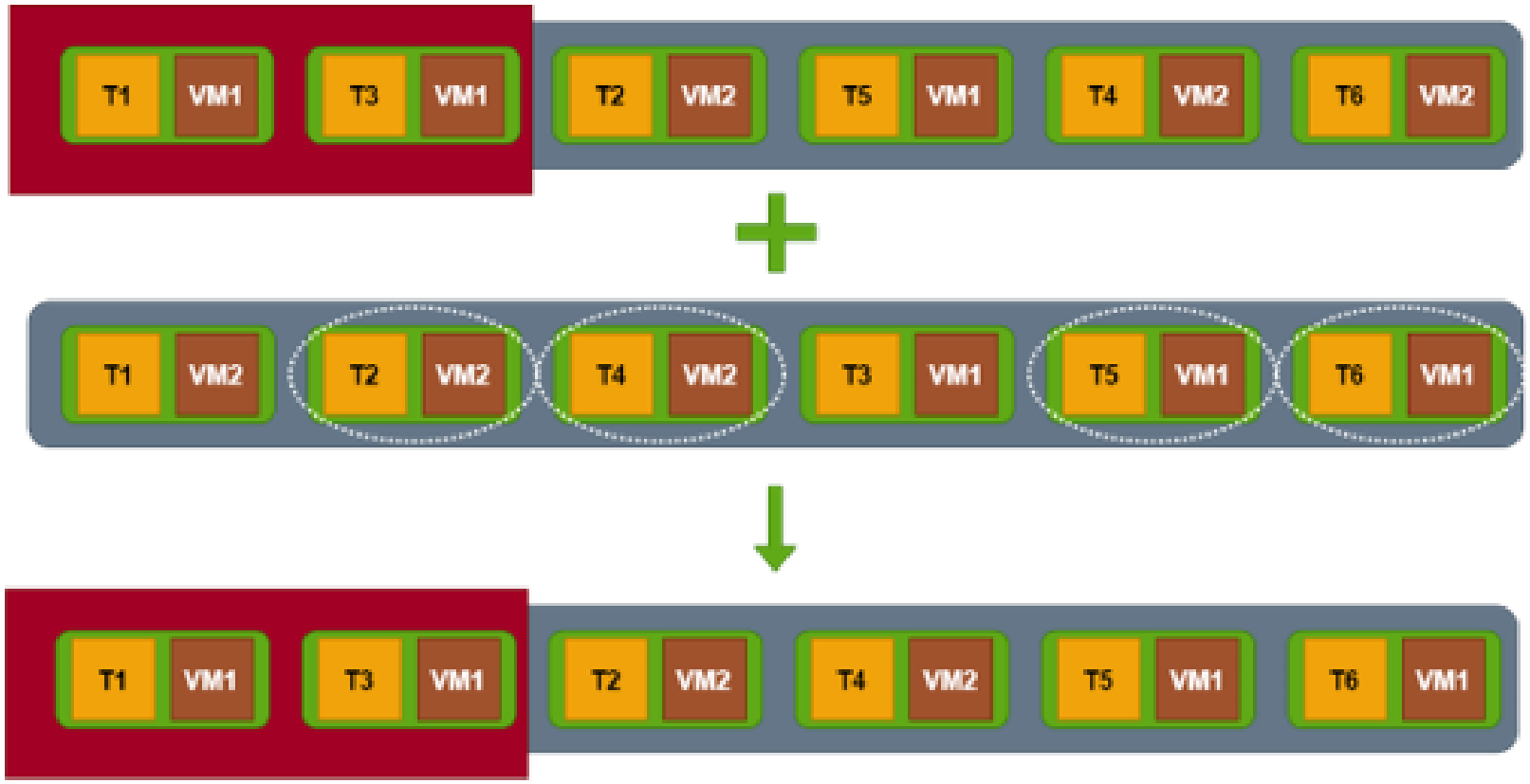




Mutación


T1
VM1
T2
VM1
T3
VM2
T5
VM1
T4
VM2
T6
VM2
T1
VM1
T2
VM1
T3
VM2
T4
VM1
T5
VM1
T6
VM2

Cruce

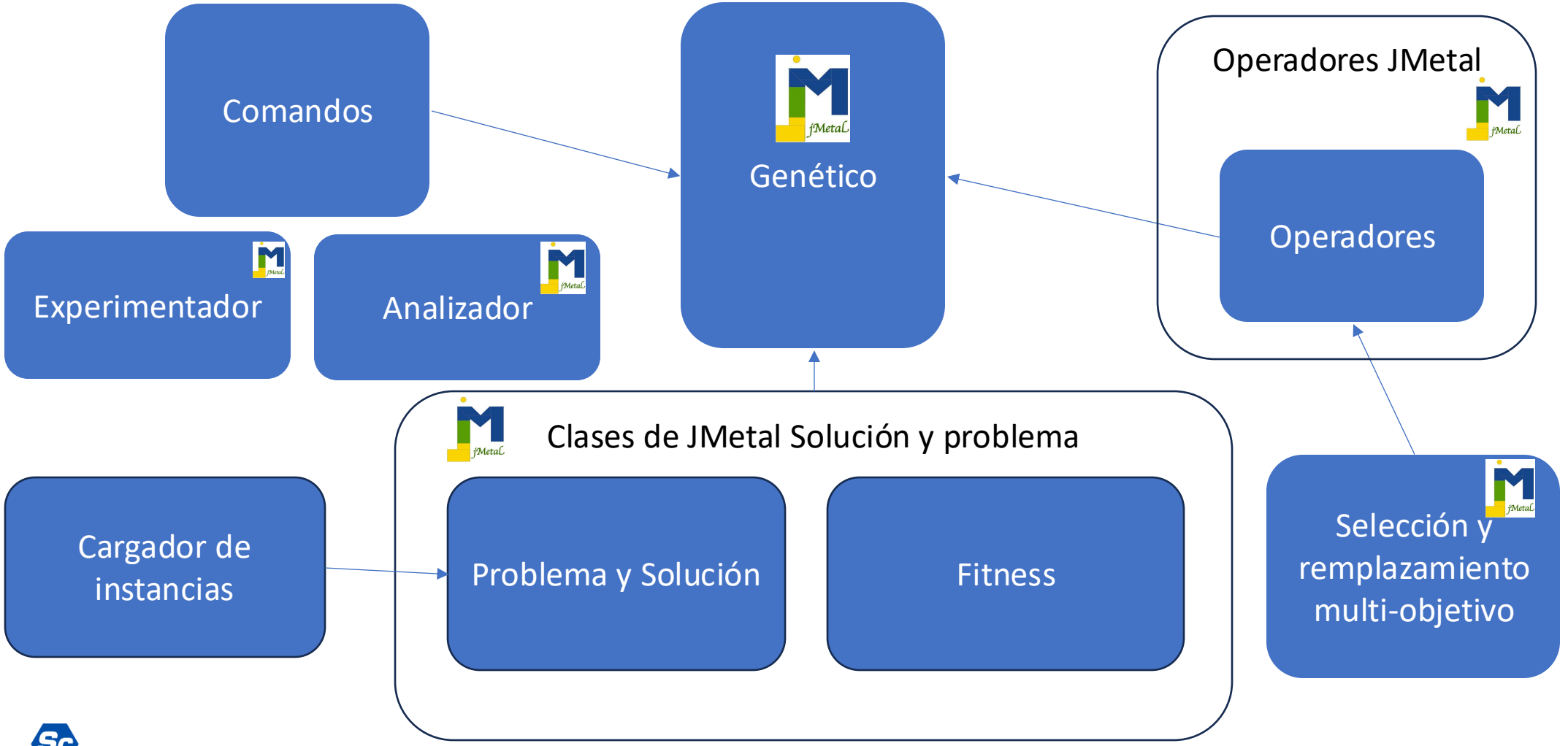

T1
VM1
T3
VM1
T2
VM2
T5
VM1
T4
VM2
T6
VM2
T1
VM2
T2
VM2
T4
VM2
T3
VM1
T5
VM1
T6
VM1
T1
VM1
T3
VM1
T2
VM2
T4
VM2
T5
VM1
T6
VM1

Codificando la solución

Porqué usar un Framework

- Ofrece distintos operadores.
- Cuenta con una librería *de problemas clásicos.*
- *Contiene la implementación de múltiples algoritmos mono-objetivo y multi-objetivo.*
- *Ofrece una suite de experimentación y analisis.*

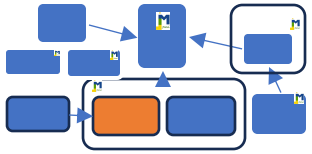
Organización del proyecto



Problema y Solución: Task

Task.java | TaskFile.java | FileList.java | Host.java | InstanceData.java | BenchmarkData.java

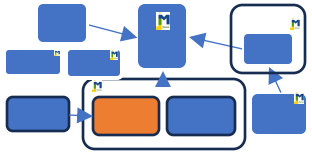
```java
/** Contine la información de una tarea. */
public class Task {

  private String name;
  private Double runtime;
  private List<Task> parents;
  private List<Task> children;
  private FileList input;
  private FileList output;
}
```



Problema y Solución: TaskFile

Task.java | TaskFile.java | FileList.java | Host.java | InstanceData.java | BenchmarkData.java

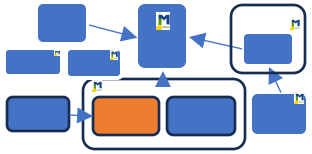
```java
/** Representa el archivo de una tarea. */
public class TaskFile {

  private String name;
  private Direction link;
  private Long size;
}
```



Problema y Solución: FileList

Task.java | TaskFile.java | FileList.java | Host.java | InstanceData.java | BenchmarkData.java

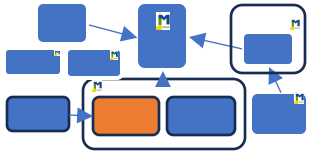
```java
/** Contine la información de los archivos en una tarea. */
public class FileList {
  private List<TaskFile> files;
  private Long sizeInBits;
}
```



Problema y Solución: Host

Task.java | TaskFile.java | FileList.java | Host.java | InstanceData.java | BenchmarkData.java

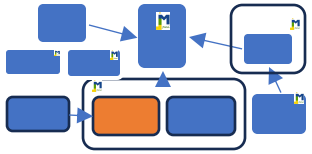
```java
/** Definición de un servidor. */
public class Host {

  private final String name;
  private final Long flops;
  private final Long diskSpeed;
  private final Long networkSpeed;
  private final Double energyCost;
  private final Double energyCostStandBy;

}
```



Problema y Solución: InstanceData

Task.java | TaskFile.java | FileList.java | Host.java | InstanceData.java | BenchmarkData.java

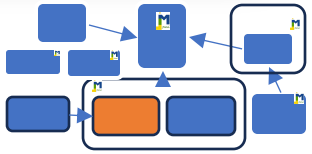
```java
/**
 * Estructura con la información de la instancia a resolver.
 *
 * @param workflow Lista de tareas identificadas por su ID.
 * @param hosts Lista de máquinas virtuales indentificadas por su ID.
 * @param referenceFlops Velocidad del procesador utilizado en el workflow original.
 */
public record InstanceData(Map<String, Task> workflow, Map<String, Host> hosts, Long referenceFlops)
    implements Serializable {}
```



Problema y Solución: BenchmarkData

Task.java | TaskFile.java | FileList.java | Host.java | InstanceData.java | BenchmarkData.java

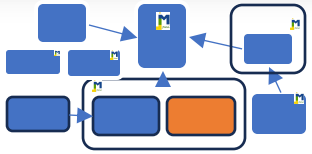
```java
/**
 * Contiene la información de una ejecución.
 *
 * @param workflow El workflow ejecutado.
 * @param hosts El numero de hosts utilizados.
 * @param fitness La función de fitness utilizada.
 * @param makespan El makespan.
 * @param energy La energia consumida.
 * @param time El tiempo necesario de ejecución del algoritmo genético.
 */
public record BenchmarkData(
    String workflow, Integer hosts, String fitness, Double makespan, Double energy, Long time) {}
```



Fitness: TaskSchedule

TaskSchedule.java | FitnessInfo.java | TaskCosts.java | ParentsInfo.java | PlanPair.java

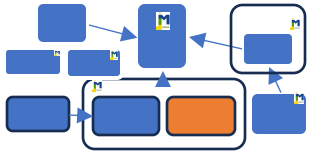
```java
/**
 * Planificación para cada tarea.
 *
 * @param task La tarea que ha de ser planificada.
 * @param ast Cuando comienza en la linea temporal.
 * @param eft Cuando termina en la linea temporal.
 * @param host En que máquina virtual se ejecuta la tarea.
 */
public record TaskSchedule(Task task, Double ast, Double eft, Host host) {}
```



Fitness: FitnessInfo

TaskSchedule.java | FitnessInfo.java | TaskCosts.java | ParentsInfo.java | PlanPair.java

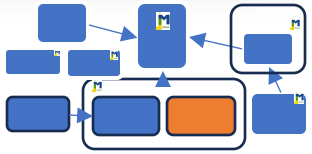
```java
/**
 * Contiene la información relativa al fitness de una planificación.
 *
 * @param fitness Lista de los valores de los distintos objetivos (Makespan y Energia).
 * @param schedule La planificación.
 * @param fitnessFunction Función de planificación utilizada.
 */
public record FitnessInfo(Map<String, Double> fitness, List<TaskSchedule> schedule, String fitnessFunction) {}
```



Fitness: TaskCosts

TaskSchedule.java | FitnessInfo.java | TaskCosts.java | ParentsInfo.java | PlanPair.java

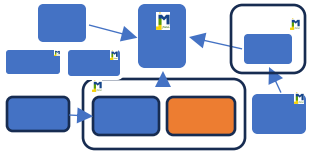
```java
/**
 * El coste de ejecutar una tarea
 *
 * @param diskReadStaging Tiempo necesario para leer los archivos de entrada iniciales.
 * @param diskWrite Tiempo necesario para escribir los archivos a disco.
 * @param eft En que segundo de la linea temporal termina la tarea.
 * @param taskCommunications Tiempo necesario para obtener todos los archivos de los padres.
 */
public record TaskCosts(
    Double diskReadStaging, Double diskWrite, Double eft, Double taskCommunications, Double ast) {}
```



Fitness: ParentsInfo

TaskSchedule.java | FitnessInfo.java | TaskCosts.java | ParentsInfo.java | PlanPair.java

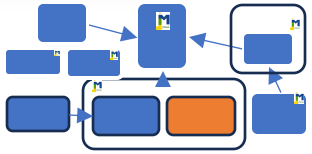
```java
/**
 * Información relativa a los padres.
 *
 * @param maxEst El ultimo tiempo de finalización de los padres.
 * @param taskCommunications Tiempo necesario para tranferir todos los datos de los padres.
 */
public record ParentsInfo(Double maxEst, Double taskCommunications) {}
```



Fitness: PlanPair

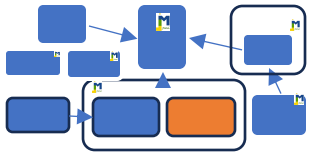
TaskSchedule.java | FitnessInfo.java | TaskCosts.java | ParentsInfo.java | PlanPair.java

```java
/**
 * Representa el par Tarea y MV contenido en una planificación.
 *
 * @param task Tarea a ser planificada.
 * @param host MV que ejecutará la tarea.
 */
public record PlanPair(Task task, Host host) {

  @Override
  public String toString() {
    return "[" + task + ", " + host + ']';
  }
}
```



FitnessCalculator.java | FitnessCalculatorSimple.java | FitnessCalculatorHeft.java

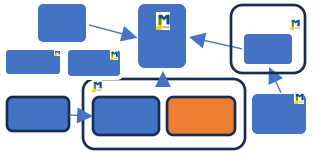
```java
public abstract class FitnessCalculator {

  private InstanceData instanceData;
  private Map<String, Map<String, Double>> computationMatrix;
  private Map<String, Map<String, Long>> networkMatrix;
  // ...

  private static Map.Entry<Task, Map<String, Long>> calculateTasksCommns(Task task) {
    // Calculamos todos los archivos a tranferir de una tarea.
  }
  public static FitnessCalculator getFitness(String fitness, InstanceData instanceData) {
    // Dado el nombre de un fitness obtenemos su implementación.
  }

  public abstract FitnessInfo calculateFitness(SchedulePermutationSolution solution);

  public TaskCosts calculateEftSemiActive(
    Task task, Host host, Map<String, TaskSchedule> schedule, Map<String, Double> available) {
    // Calculamos el tiempo de finalización de una tarea sin inserción.
  }

  public TaskCosts calculateEftActive(Task task, Host host, Map<String, TaskSchedule> schedule,
    Map<String, List<ScheduleGap>> available) {
    // Calculamos el tiempo de finalizaci'on de una tarea rellenando huecos.
  }

  public ParentsInfo findTaskCommunications(Task task, Host host, Map<String, TaskSchedule> schedule) {
    // Calcula el tiempo empleado en obtener los ficheros y cuando podria empezar la tarea como pronto
    // en función de los padres.
  }

  // ...
}
```



Fitness: FitnessCalculatorSimple

FitnessCalculator.java | FitnessCalculatorSimple.java | FitnessCalculatorHeft.java

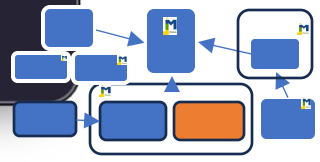
```java
public class FitnessCalculatorSimple extends FitnessCalculator {

  public FitnessInfo calculateFitness(SchedulePermutationSolution solution) {
    // Implementación del fitness más basico en el que se calcula dada una solución,
    // el makespan y energía sin modificar el chromosoma.
  }

}
```



Fitness: FitnessCalculatorHeft

FitnessCalculator.java | FitnessCalculatorSimple.java | FitnessCalculatorHeft.java

```java
public class FitnessCalculatorHeft extends FitnessCalculator {

  public FitnessInfo calculateFitness(SchedulePermutationSolution solution) {
    // Implementación del fitness aplicando la segunda parte del algoritmo Heft, se calcula
    // el makespan y energía modificando el chromosoma.

    // Por cada tarea miramos de planificarla en cada una de las máquinas usando una función
    // auxiliar, escogemos la máquina que reporte el menor EFT.
  }

  private EftAndAst calculateHeftTaskCost(
    Task task, HashMap<String, TaskSchedule> schedule, Map<String, List<ScheduleGap>> available) {
    // Calculamos el tiempo de inicio y finalización de una tarea con un algoritmo de inserción.
  }
}
```



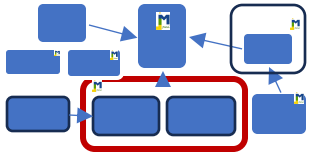
Añadiendo JMetal

- Creamos las clases de problema y solución heredando de las de Jmetal
- Creamos los operadores
- Creamos un objeto algoritmo con el genético que queramos

JMetal: SchedulingProblem

SchedulingProblem.java | SchedulePermutationSolution.java

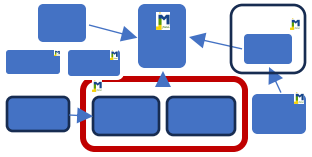
```java
public class SchedulingProblem implements PermutationProblem<SchedulePermutationSolution> {

  private final WorkflowLoader workflowLoader;
  private final HostLoader hostLoader;
  private final PlanGenerator planGenerator;

  private final FitnessCalculator fitnessCalculator;
  private InstanceData instanceData;
  private String name;
  private List<Objective> objectives;


  @Override
  public SchedulePermutationSolution createSolution() {
    var plan = planGenerator.generatePlan();
    return new SchedulePermutationSolution(numberOfVariables(), numberOfObjectives(), plan);
  }

  @Override
  public SchedulePermutationSolution evaluate(
      SchedulePermutationSolution schedulePermutationSolution) {
    /// Evaluamos la solución y actualizamos los objetivos.
    ///  ...
  }
}
```



JMetal: SchedulePermutationSolution

SchedulingProblem.java | SchedulePermutationSolution.java

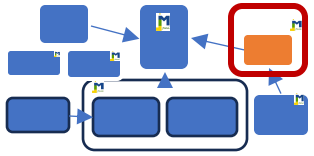
```java
public class SchedulePermutationSolution extends AbstractSolution<PlanPair>
    implements PermutationSolution<PlanPair> {

  FitnessInfo fitnessInfo;
  List<PlanPair> plan;


  @Override
  public Solution<PlanPair> copy() {
    // Copia la estructura de datos.
  }

  @Override
  public List<PlanPair> variables() {
    return plan;
  }
}
```



Operadores: ScheduleCrossover

ScheduleCrossover.java | ScheduleMutation.java | ScheduleSelection.java | ScheduleReplacement.java

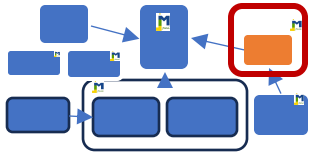
```java
public class ScheduleCrossover implements CrossoverOperator<SchedulePermutationSolution> {

  private final Operators operators;
  double crossoverProbability;

  @Override
  public int numberOfRequiredParents() {
    return 2;
  }

  @Override
  public int numberOfGeneratedChildren() {
    return 2;
  }

  @Override
  public List<SchedulePermutationSolution> execute(
      List<SchedulePermutationSolution> schedulePermutationSolutions) {
    // Dada la lista de los padres, genera dos nuevos hijos aplicando el cruce en un punto.
  }
}
```



Operadores: ScheduleMutation

ScheduleCrossover.java | ScheduleMutation.java | ScheduleSelection.java | ScheduleReplacement.java

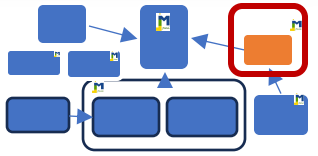
```java
public class ScheduleMutation implements MutationOperator<SchedulePermutationSolution> {
  final Operators operators;
  double mutationProbability;

  @Override
  public SchedulePermutationSolution execute(
      SchedulePermutationSolution schedulePermutationSolution) {

    return new SchedulePermutationSolution(
        schedulePermutationSolution.variables().size(),
        schedulePermutationSolution.objectives().length,
        operators.mutate(schedulePermutationSolution.getPlan()),
        schedulePermutationSolution.getArbiter());
  }
}
```



Operadores: ScheduleSelection

ScheduleCrossover.java | ScheduleMutation.java | ScheduleSelection.java | ScheduleReplacement.java

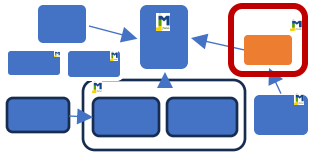
```java
public class ScheduleSelection implements Selection<SchedulePermutationSolution> {
  Random random;

  @Override
  public List<SchedulePermutationSolution> select(List<SchedulePermutationSolution> list) {
    var listToShuffle = new ArrayList<>(list);
    Collections.shuffle(listToShuffle, random);
    return listToShuffle;
  }
}
```



Operadores: ScheduleReplacement

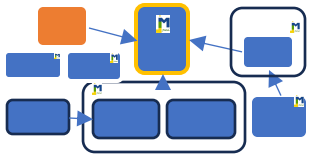
ScheduleCrossover.java | ScheduleMutation.java | ScheduleSelection.java | ScheduleReplacement.java

```java
public class ScheduleReplacement implements Replacement<SchedulePermutationSolution> {

  @Override
  public List<SchedulePermutationSolution> replace(
      List<SchedulePermutationSolution> parents, List<SchedulePermutationSolution> children) {
    // Se realiza un torneo 4:2, de 4 posibles soluciones se escogen las 2 mejores.
  }
}
```



EvaluateCommand.java | ExperimentCommand.java | ExperimentJmetalCommand.java

```java
@Command
public class EvaluateCommand {
  final WorkflowLoader workflowLoader;
  final HostLoader hostLoader;

  final ScheduleExporter scheduleExporter;

  @Command(command = "evaluate")
  public String evaluate(
      @Option(shortNames = 'H', required = true) String hostsFile,
      @Option(shortNames = 'W', required = true) String workflowFile,
      @Option(shortNames = 'E', defaultValue = "1000") Integer executions,
      @Option(shortNames = 'S', defaultValue = "1") Long seed,
      @Option(shortNames = 'F', defaultValue = "simple") String fitness) {

    // Cargamos el problema, configuramos los operadores, fijamos la población inicial
    // definimos el criterio de parada.

    // ...

    // Creamos el algoritmo evolutivo.
    EvolutionaryAlgorithm<SchedulePermutationSolution> gaAlgo =
          new GeneticAlgorithmBuilder<>(
                  "GGA", problem, populationSize, offspringPopulationSize, crossover, mutation)
              .setTermination(termination)
              .setEvaluation(new MultiThreadedEvaluation<>(0, problem))
              .setSelection(new ScheduleSelection(random))
              .setReplacement(new ScheduleReplacement(random, objectives.get(0)))
              .build();
    gaAlgo.run();
    var population = gaAlgo.result();
    printFinalSolutionSet(population);
  }
}
```



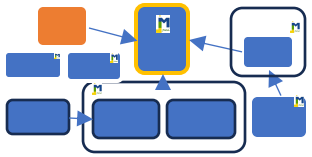
Plan de experimentación

- Definir las instancias
- Escoger la infraestructura
- Definir los parámetros

Comandos: ExperimentConfig

experimentConfig.json

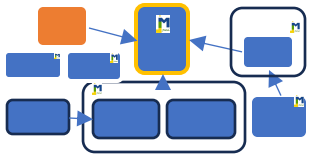
```
{
  "fitness": [
    "heft-energy-mono-active", "simple-energy-mono",
    "min-energy-UM-mono-active", "fvlt-me-mono-active"
  ],
  "workflows": [
    "1000genome-chameleon-2ch-250k-001",
    "1000genome-chameleon-4ch-250k-001",
    "cycles-chameleon-1l-1c-9p-001",
    "cycles-chameleon-2l-1c-9p-001",
    "epigenomics-chameleon-hep-1seq-100k-001",
    "epigenomics-chameleon-hep-6seq-100k-001",
  ],
  "minHosts": 2,
  "maxHosts": 16,
  "hostIncrement": 2,
  "referenceSpeed": "121.6Gf",
  "independentRuns": 10,
  "objectives": [
    "energy",
    "makespan"
  ]
}
```



Comandos: ExperimentCommand

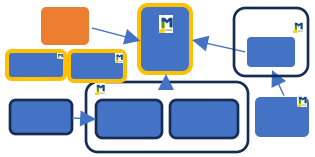
EvaluateCommand.java | ExperimentCommand.java | ExperimentJmetalCommand.java

```java
@Command
public class ExperimentCommand {
  final WorkflowLoader workflowLoader;
  final HostLoader hostLoader;
  final ExperimentConfigLoader experimentConfigLoader;

    @Command(command = "experiment")
  public String experiment(
      @Option(shortNames = 'W') String workflowsPath,
      @Option(shortNames = 'H') String hostsPath,
      @Option(shortNames = 'T') String type,
      @Option(shortNames = 'E', defaultValue = "100000") Integer executions,
      @Option(shortNames = 'S', defaultValue = "1") Long seed,
      @Option(shortNames = 'C') String experimentConfigFile) {
    // Realiza la sesión de experimentos para cada uno de los workflows, con
    // la distintas configuraciones de host. Realizamos al final el calculo de
    // las medias de cada ejecución independiente.

    // Creamos una tabla excel con toda la información para su posterior analisis.
  }
}
```



EvaluateCommand.java | ExperimentCommand.java | ExperimentJmetalCommand.java

```java
@Command
public class ExperimentJmetalCommand {

  @Command(command = "jmetal")
  public String experiment(
      @Option(shortNames = 'W') String workflowsPath,
      @Option(shortNames = 'H') String hostsPath,
      @Option(shortNames = 'T') String type,
      @Option(shortNames = 'E', defaultValue = "100000") Integer executions,
      @Option(shortNames = 'S', defaultValue = "1") Long seed,
      @Option(shortNames = 'C') String experimentConfigFile) {

        // Obtenemos la lista de problemas y algoritmos.
        // ...

        // Generamos el plan de experimentación.
        Experiment<SchedulePermutationSolution, List<SchedulePermutationSolution>> experiment =
        new ExperimentBuilder<SchedulePermutationSolution, List<SchedulePermutationSolution>>(
                "Scheduling")
            .setAlgorithmList(algorithmList)
            .setProblemList(problemList)
            .setExperimentBaseDirectory(experimentBaseDirectory)
            .setOutputParetoFrontFileName("FUN")
            .setOutputParetoSetFileName("VAR")
            .setReferenceFrontDirectory(experimentBaseDirectory + "/Scheduling/referenceFronts")
            .setIndicatorList(
                List.of(
                    new Epsilon(),
                    new Spread(),
                    new GenerationalDistance(),
                    new PISAHypervolume()))
            .setIndependentRuns(experimentConfig.independentRuns())
            .build();
    new ExecuteAlgorithms<>(experiment).run();

    // Computamos todas las metricas e informes con las funciones de Jmetal.

  }
}
```



Ejemplo de salida

HV

Median values

	heft-energy-mono	simple-mono	multi-energy-mono
1000genome-chameleon-2ch-250k-001-hosts-16	3.92201E-2	7.56931E-16	5.70401E-2

Wilcoxon Test

	simple-mono	multi-energy-mono
heft-energy-mono	+	=
simple-mono		-

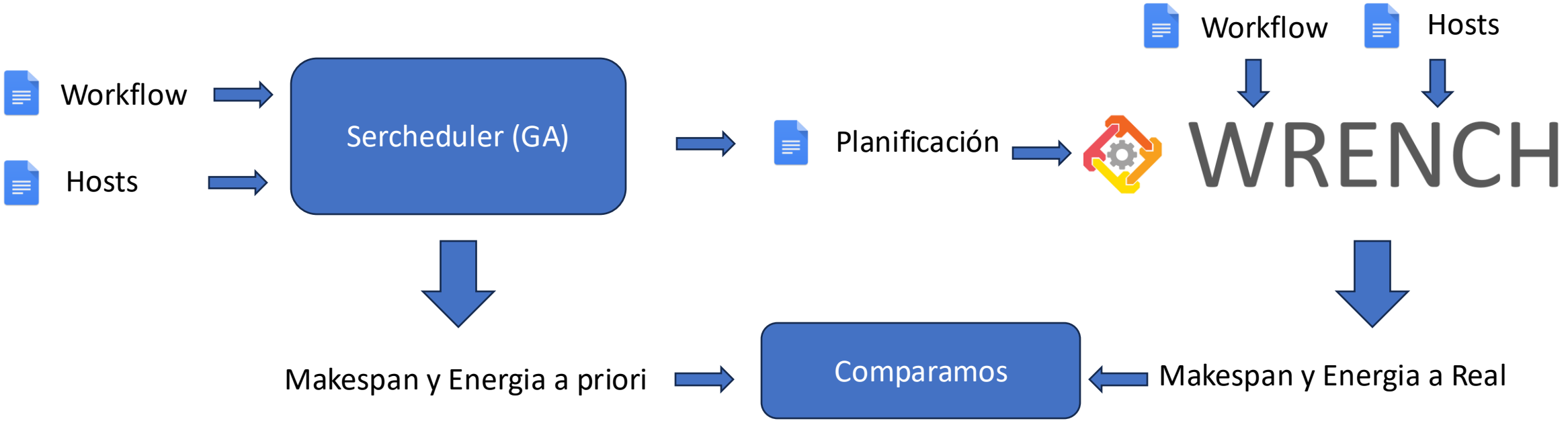
Friedman ranking and Holm test

Algorithm	Ranking	p-value	Holm	Hypothesis
multi-energy-mono	1	0E0	0	-
heft-energy-mono	2	1.075E-4	0.05	Rejected
simple-mono	3	9.486E-15	0.025	Rejected

Comprobar los resultados

- En los problemas en los que se hagan simplificaciones es necesario verificar los datos con un simulador.
- Se pueden repetir los análisis de JMetal sustituyendo los valores a priori por los simulados.

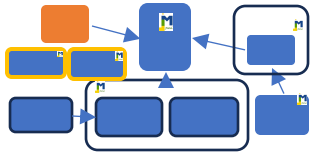
Ejecutando en el Simulador



Comandos: SimulateCommand

SimulateCommand.java

```java
@Command
public class SimulateCommand {
  static final Logger LOG = LoggerFactory.getLogger(SimulateCommand.class);
  private final ExperimentParser experimentParser;
  private final HostLoader hostLoader;
  private final ExperimentConfigLoader experimentConfigLoader;

  @Command(command = "simulate")
  public String simulate(@Option(shortNames = 'C') String experimentConfigFile) {

    var experimentConfig = experimentConfigLoader.readFromFile(new File(experimentConfigFile));
    var benchmarks = experimentConfig.workflows();

    var hostsWrench = loadHostsWrench();

    for (var benchmark : benchmarks) {

      for (var fitness : experimentConfig.fitness()) {

        for (int i = 0; i < experimentConfig.independentRuns(); i++) {
          SimulateExperiment(benchmark, fitness, i, hostsWrench);
        }
      }
    }
}
```



En resumen

- Hacer un análisis del problema y sus componentes.
- Codificando la solución de manera iterativa.
- Definir un plan de experimentación.
- Analizar los resultados.
- Se puede resolver cualquier instancia no conocida.

Dudas y preguntas

- Preguntar aquí.
- Preguntarme a la salida.
- Enviarme un mail a uo237136@uniovi.es.
- Contactar conmigo por linkedin: Pablo Barredo Gil.