# GB-500: Introdução a Workflows Científicos e suas Aplicações

Professores: Luiz Gadelha e Kary Ocaña

Programa de Pós-Graduação em Modelagem Computacional, P3/2015
Laboratório Nacional de Computação Científica

23 de junho de 2015

Laboratório Nacional de Computação Científica

# Referências


- Workflows Científicos:
  - Braguetto, K., Cordeiro, D. (2014). Introdução à Modelagem e Execução de Workflows Científicos. In XXXIII Jornadas de Atualização em Informática (JAI). CSBC 2014.
  - Cuevas-Vicenttín, V. et al. (2012). Scientific Workflows and Provenance: Introduction and Research Opportunities. Datenbank-Spektrum, 12(3), 193–203.
  - E. Deelman et al. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems* 25(1):528-540, 2009.
  - Liu, J. et al. (2015). A Survey of Data-Intensive Scientific Workflow Management. Journal of Grid Computing.
  - Taylor, I. et al. (2007). Workflows for e-Science: Scientific Workflows for Grids. Springer.

# Referências


- Proveniência:
  - Carata, L. et al. (2014). A primer on provenance. Communications of the ACM, 57(5), 52–60.
  - Miles, S. et al. (2011). PrIMe: A Methodology for Developing Provenance-Aware Applications. ACM Transactions on Software Engineering and Methodology, 20(3), 1–42.
  - Moreau, L., Groth, P. (2013). Provenance: An Introduction to PROV. Synthesis Lectures on the Semantic Web: Theory and Technology (Vol. 3). Morgan & Claypool.

# Workflows Científicos

- Um **workflow científico** consiste da especificação de um conjunto de aplicações científicas a serem executadas e suas dependências mútuas.
- Segue um ciclo de vida análogo ao dos experimentos científicos computacionais:
  - **Composição, representação e modelagem de dados**.
  - Mapeamento e execução.
  - Coleta de metadados e proveniência.
- Um **sistema de gerência de workflows científicos** (SGWC) permite gerenciar o ciclo de vida de workflows científicos.

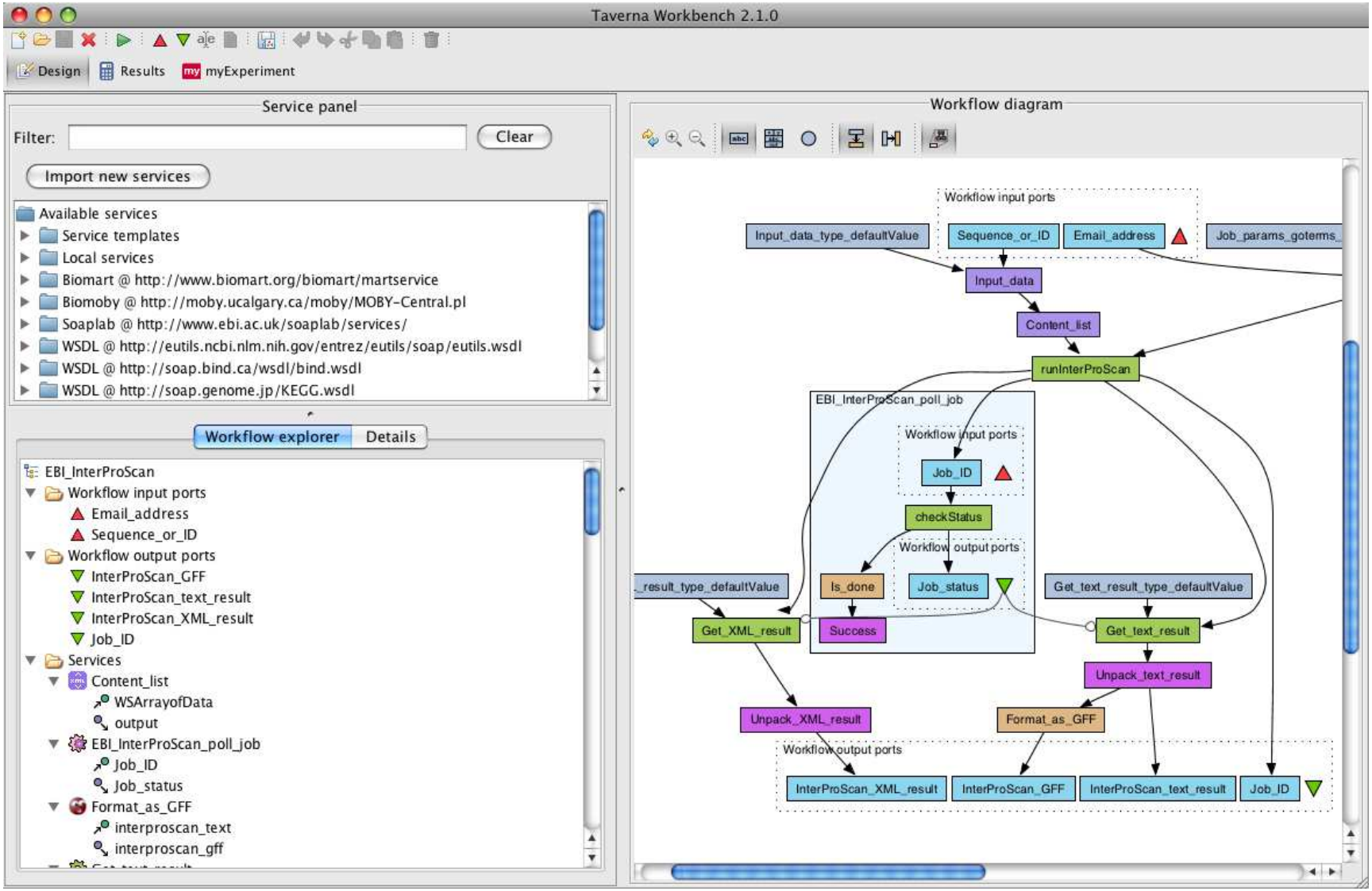
# Composição de Workflows Científicos

- Etapa em que são definidas as aplicações componentes e as dependências de dados.
- Níveis de especificação:
  - Abstrato: tarefas abstratas, descritas por funcionalidade geral (p. ex., comparação de seqüências).
  - Concreto: aplicações científicas e conjuntos de dados específicos.
- Representação:
  - Textual: linguagem de programação.
  - Gráfica: interface gráfica onde nós são tarefas e arestas são dependências.

# Exemplo: Composição Textual

```
type fastaseq;
type headerfile;
type indexfile;
type seqfile;
type database
{
  headerfile phr;
  indexfile pin;
  seqfile psq;
}
type query;
type output;
string num_partitions=@arg("n", "8");
string program_name=@arg("p", "blastp");
fastaseq dbin <single_file_mapper;file=@arg("d", "database")>;
query query_file <single_file_mapper;file=@arg("i", "sequence.seq")>;
string expectation_value=@arg("e", "0.1");
output blast_output_file <single_file_mapper;file=@arg("o",
                                                       "output.html")>;
string filter_query_sequence=@arg("F", "F");
fastaseq partition[] <ext;exec="splitmapper.sh",n=num_partitions>;

app (fastaseq out[]) split_database (fastaseq d, string n)
{
  fastasplitn @filename(d) n;
}

app (database out) formatdb (fastaseq i)
{
  formatdb "-i" @filename(i);
}

app (output o) blastapp(query i, fastaseq d, string p, string e, string f,
                        database db)
{
  blastall "-p" p "-i" @filename(i) "-d" @filename(d) "-o" @filename(o)
           "-e" e "-T" "-F" f;
}

app (output o) blastmerge(output o_frags[])
{
  blastmerge @filename(o) @filenames(o_frags);
}

partition=split_database(dbin, num_partitions);

database formatdbout[] <ext; exec="formatdbmapper.sh",n=num_partitions>;
output out[] <ext; exec="outputmapper.sh",n=num_partitions>;

foreach part,i in partition {
  formatdbout[i] = formatdb(part);
  out[i]=blastapp(query_file, part, program_name, expectation_value,
                  filter_query_sequence, formatdbout[i]);
}

blast_output_file=blastmerge(out);
```

# Exemplo: Composição Gráfica

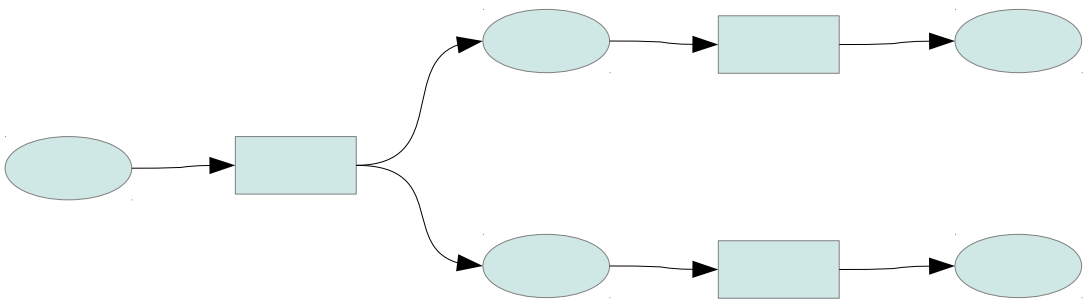



Fonte: Taverna (`http://www.taverna.org.uk`)

# Composição de Workflows Científicos: Padrões

- Foram identificados 43 padrões de composição de workflows.
- Exemplos:
  - Seqüência:

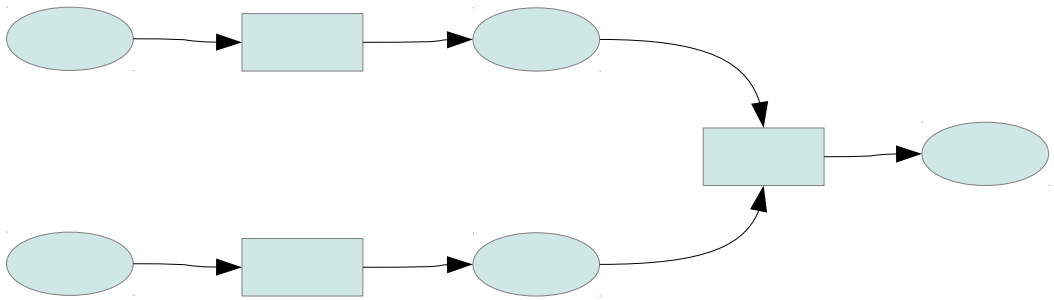
  - Bifurcação paralela:



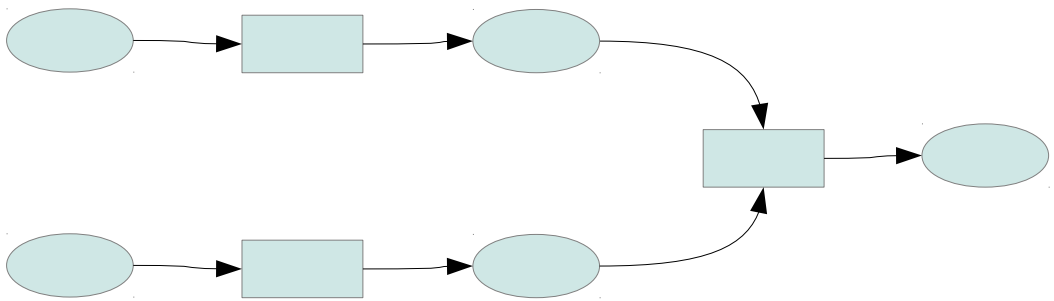

- W. van der Aalst et al. Workflow Patterns. *Distributed and Parallel Databases* 14(1):5-51, 2003.
- N. Russell et al. Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, 2006.
- Workflow Patterns (http://www.workflowpatterns.com).

# Composição de Workflows Científicos: Padrões

- Exemplos:
  - Sincronização:

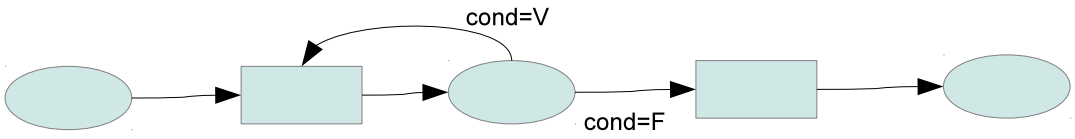


  - Fusão:

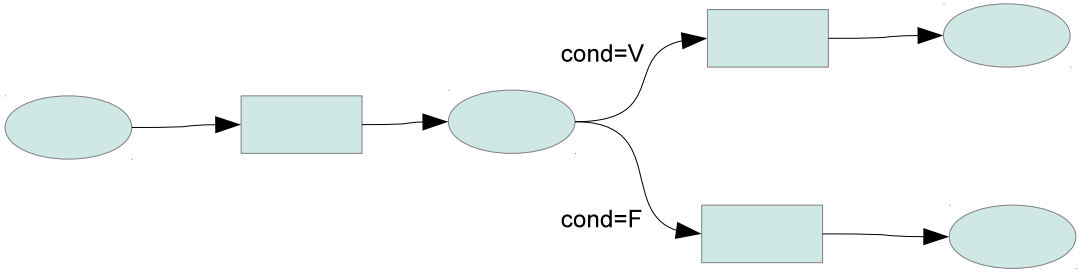


# Composição de Workflows Científicos: Padrões

- Exemplos:
  - Laço:




  - Escolha exclusiva:



# Swift

- Swift permite gerenciar workflows científicos em ambientes paralelos e distribuídos.
- Ambiente ideal para experimentação pois demonstradamente escala para 116.000 CPUs.
- É composto por:
  - Uma linguagem de alto nível para especificação de workflows científicos como scripts.
  - Ambiente de execução com suporte nativo a:
    - execução local,
    - execução paralela (p.ex. PBS, SGE),
    - execução distribuída (p.ex. Condor, Globus).
  - Sistema de gerência de proveniência.

M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford I. Raicu, Parallel Scripting for Applications at the Petascale and Beyond. *IEEE Computer*, 42(11):50-60, 2009.

M. Wilde, M. Hategan, J. Wozniak, B. Clifford, D. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):634-652, 2011.

# Swift

- Paralelismo implícito:
  - Toda expressão de um script é avaliada em paralelo, exceto quando há dependências de dados.
  - $\Rightarrow$ Não é possível assumir uma ordem de execução dessas expressões.
- Independência de localidade:
  - Os scripts não expressam transferências de dados e seleção de sítios de execução.

# Swift: Linguagem de Especificação de Workflows

- Aplicações que compõem um workflow são associadas a funções (*app functions*) em um script Swift.
- Variáveis em um script Swift podem ser associadas a:
  - tipos de dados primitivos,
  - dados persistentes armazenados em arquivos,
  - tipos de dados compostos (coleções).
- Suporte a laços e controles condicionais de fluxo.

# Swift: Linguagem de Especificação de Workflows

- Sintaxe semelhante ao C, Java.
- Estrutura geral de um script Swift:
  1. Declarações de tipos de dados.
  2. Declarações e atribuições de variáveis.
  3. Declarações e chamadas de funções associadas a aplicações.
  4. Declarações e chamadas de funções compostas.

# Swift: Linguagem de Especificação de Workflows

- Semelhanças com paradigma funcional:
  - Computação orientada a avaliação de funções.
  - Chamadas sem *efeitos colaterais*.
  - *Avaliação preguiçosa* de expressões.
  - Variáveis de *atribuição única*.

# Swift: Linguagem de Especificação de Workflows

Exemplo:

```
type messagefile;
type countfile;

app (countfile t) countwords (messagefile f) {
     wc "-w" @filename(f) stdout=@filename(t);
}

string inputNames = "foreach.1.txt foreach.2.txt foreach.3.txt";

messagefile inputfiles[] <fixed_array_mapper;files=inputNames>;

foreach f in inputfiles {
  countfile c <regexp_mapper; source=@f, match="(.*)txt",
               transform="\\1count">;
  c = countwords(f);
}
```

# Swift: Ambiente de Execução

- Execução independente de localidade.
- Paralelização automática de execuções de aplicações componentes que não tenham dependências de dados.
- Heurística de balanceamento de carga das aplicações entre os recursos disponíveis.
  - Pontuação de desempenho proporcional à taxa de execução histórica de tarefas.
- Tolerância a falhas:
  - redundância,
  - resubmissão de execuções que falharam,
  - re-execução de script sem repetição de computações.

# Swift: Ambiente de Execução

- Responsável submissão de execução de aplicações componentes e transferência de arquivos.
- Provedores de dados e de execução.
- Catálogo de aplicações componentes.
- Catálogo de recursos computacionais.

# Swift: Ambiente de Execução

- Para cada chamada a aplicação componente:
  1. verifica no catálogo de aplicações em quais recursos computacionais a aplicação está disponível,
  2. seleciona um recurso computacional (maior probabilidade para recurso com maior pontuação de desempenho),
  3. verifica no catálogo de recursos computacionais que provedor de execução e provedor de dados deve ser utilizado,
  4. se for o caso, transfere os dados para o recurso computacional,
  5. executa a aplicação componente,
  6. transfere de volta os dados resultantes.

# Swift: Ambiente de Execução

Catálogo de recursos computacionais:

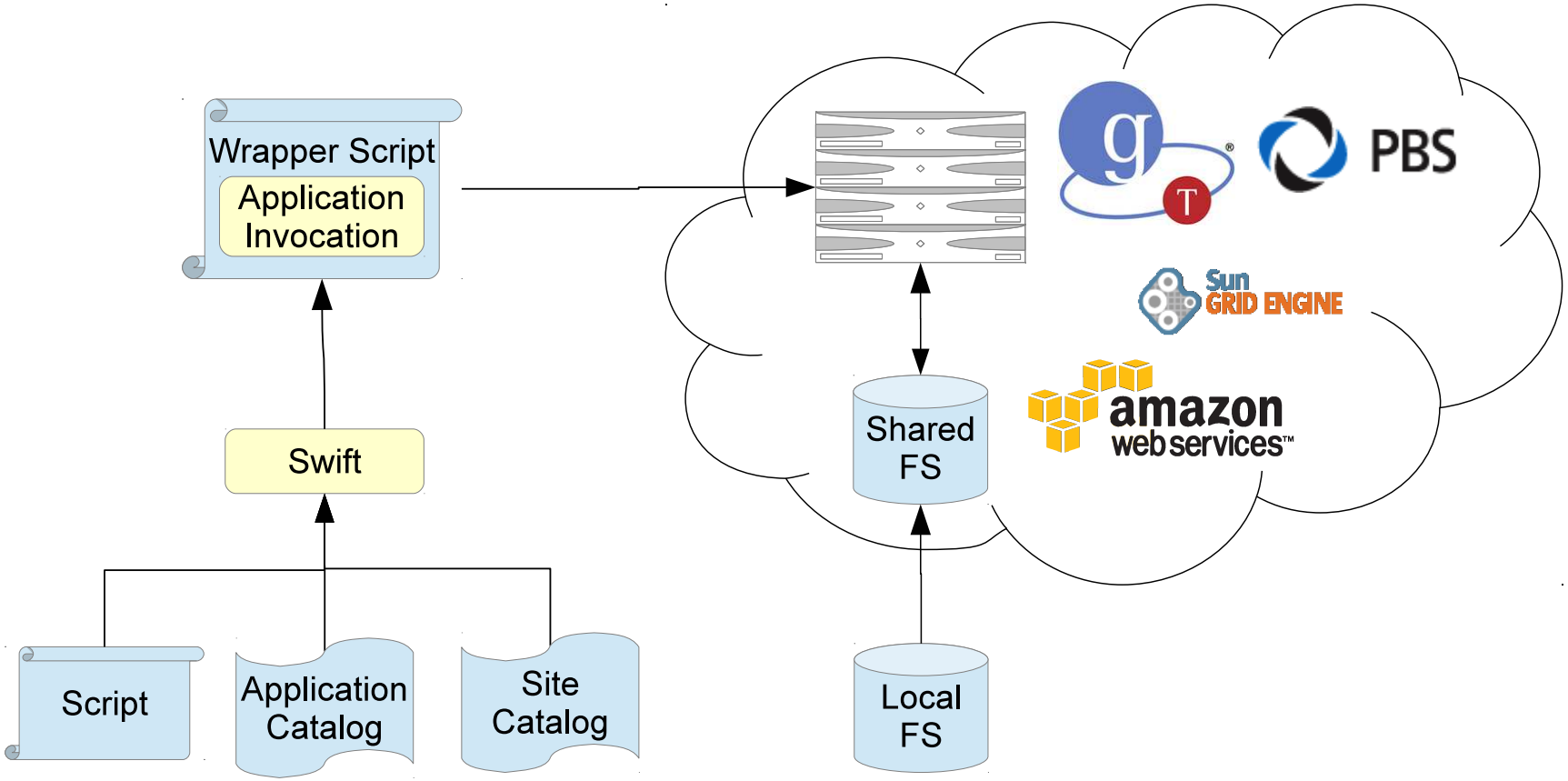
```
<config>
  <pool handle="localhost" sysinfo="INTEL32::LINUX">
    <execution provider="local" url="none" />
    <workdirectory>/prj/prjssi/lgadelha/swiftwork</workdirectory>
    <profile namespace="karajan" key="jobThrottle">0.03</profile>
  </pool>


  <pool handle="sge-local">
    <execution provider="sge" url="none" />
    <profile namespace="globus" key="pe">threads</profile>
    <profile key="jobThrottle" namespace="karajan">6.23</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <filesystem provider="local" url="none" />
    <workdirectory>/prj/prjssi/lgadelha/swiftwork</workdirectory>
  </pool>
</config>
```

# Swift: Ambiente de Execução

Catálogo de aplicações componentes:

```
localhost   echo     /bin/echo
localhost   cat      /bin/cat
localhost   ls       /bin/ls
localhost   grep     /bin/grep
localhost   sort     /bin/sort
localhost   paste    /bin/paste
localhost   cp       /bin/cp
localhost   touch    /bin/touch
localhost   wc       /usr/bin/wc
localhost   sleep    /bin/sleep
sge-local   cat      /bin/cat
```

# Swift: Ambiente de Execução



# Swift

- Desenvolvimento:



- Colaboração do LNCC no desenvolvimento do componente de gerência de proveniência: MTCProv.
- Aplicações: OOPS, SciColSim, MODIS, Labinfo.