Large-scale distributed computing systems

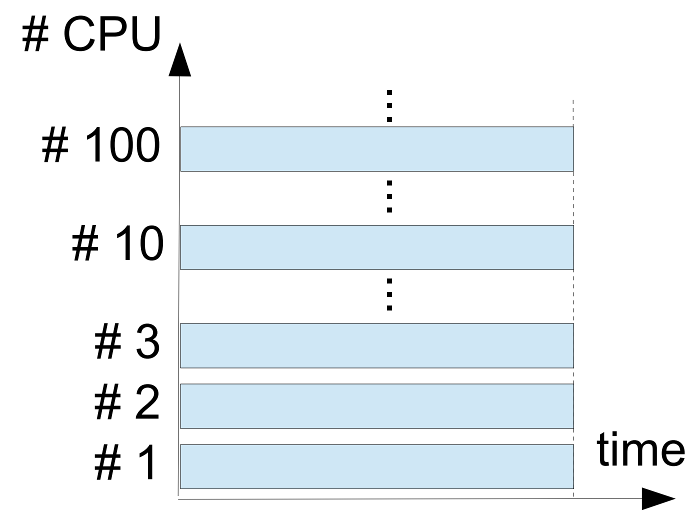
Week 3

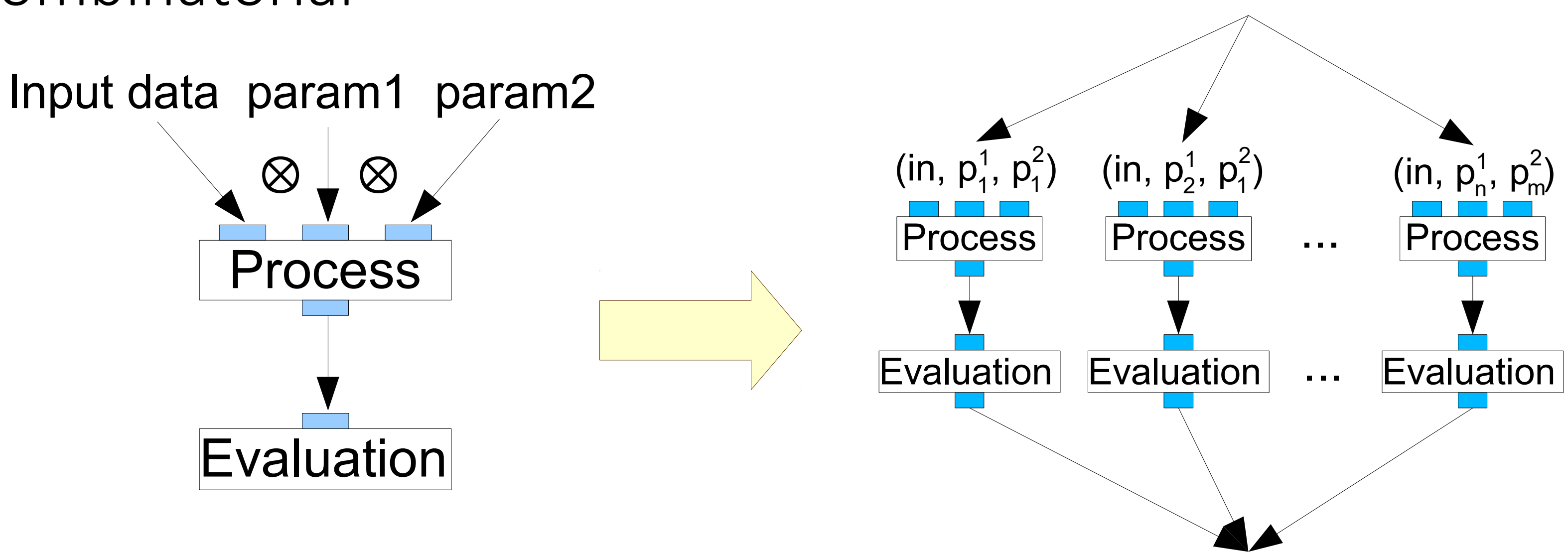
January 2017

Johan Montagnat
CNRS, I3S, SPARKS
http://www.i3s.unice.fr/~johan/

Distributed application models

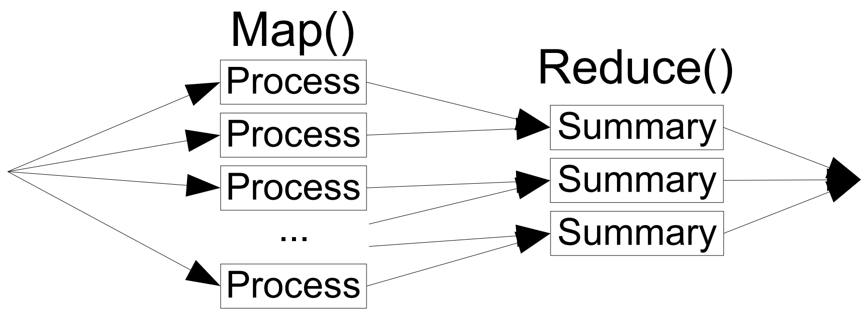
- Embarrassingly parallel applications
  - Embarrassing size
    - Scalability is the challenge
  - Trivial parallelisation
- Parameters Sweep
  - Explore process parameters space
  - Many optimization problems
  - Combinatorial



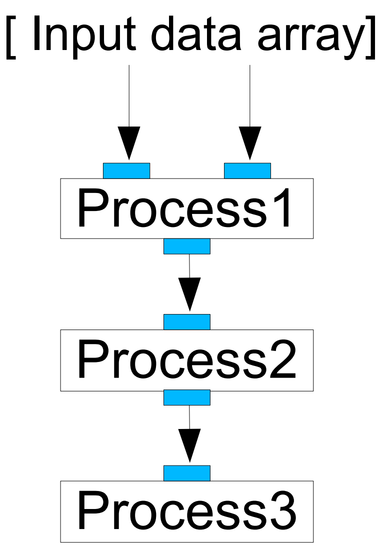


Distributed application models

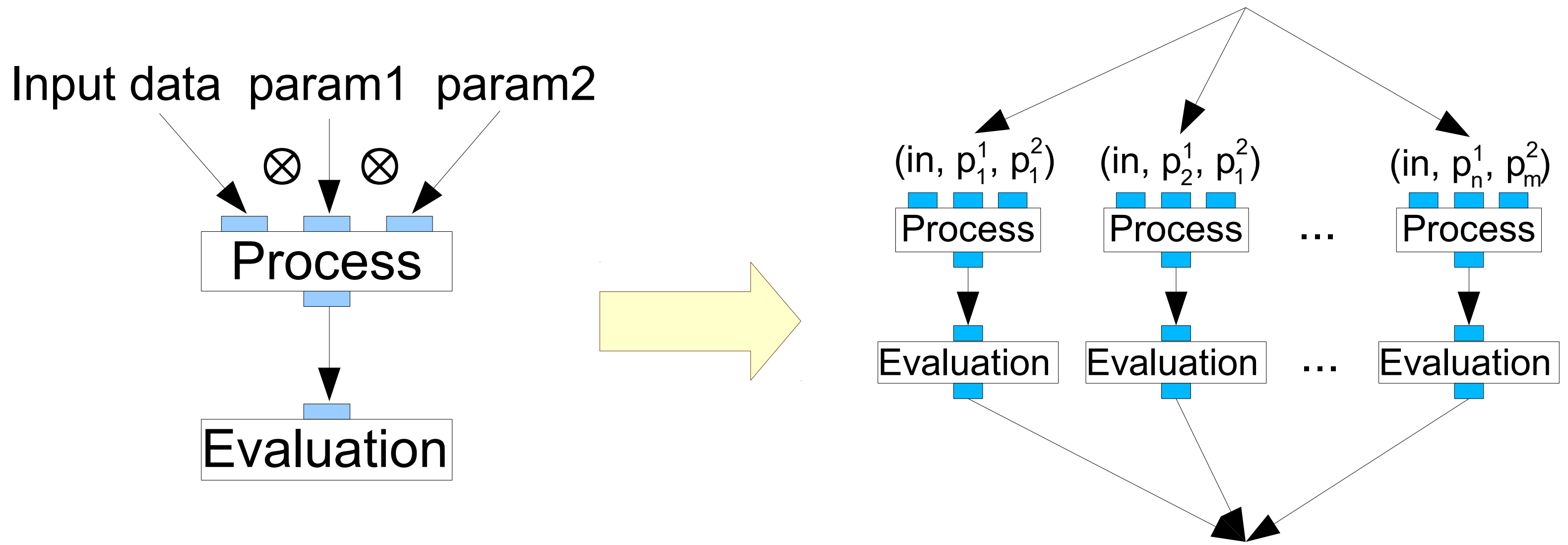
- Map-Reduce framework
  - Two steps simple parallelization framework
    - Map() function decomposes computation
    - Reduce() function combines results
  - Backed-up by fault-tolerant and scalable support tools
    - e.g. Hadoop



- SPMD (Single Processing, Multiple Data)
  - Exploit large data sets parallelism
  - Independent computations on different data items
  - Simple process: embarassingly parallel
  - Complex process: pipelines / workflows



Workflows



High-level (user) view

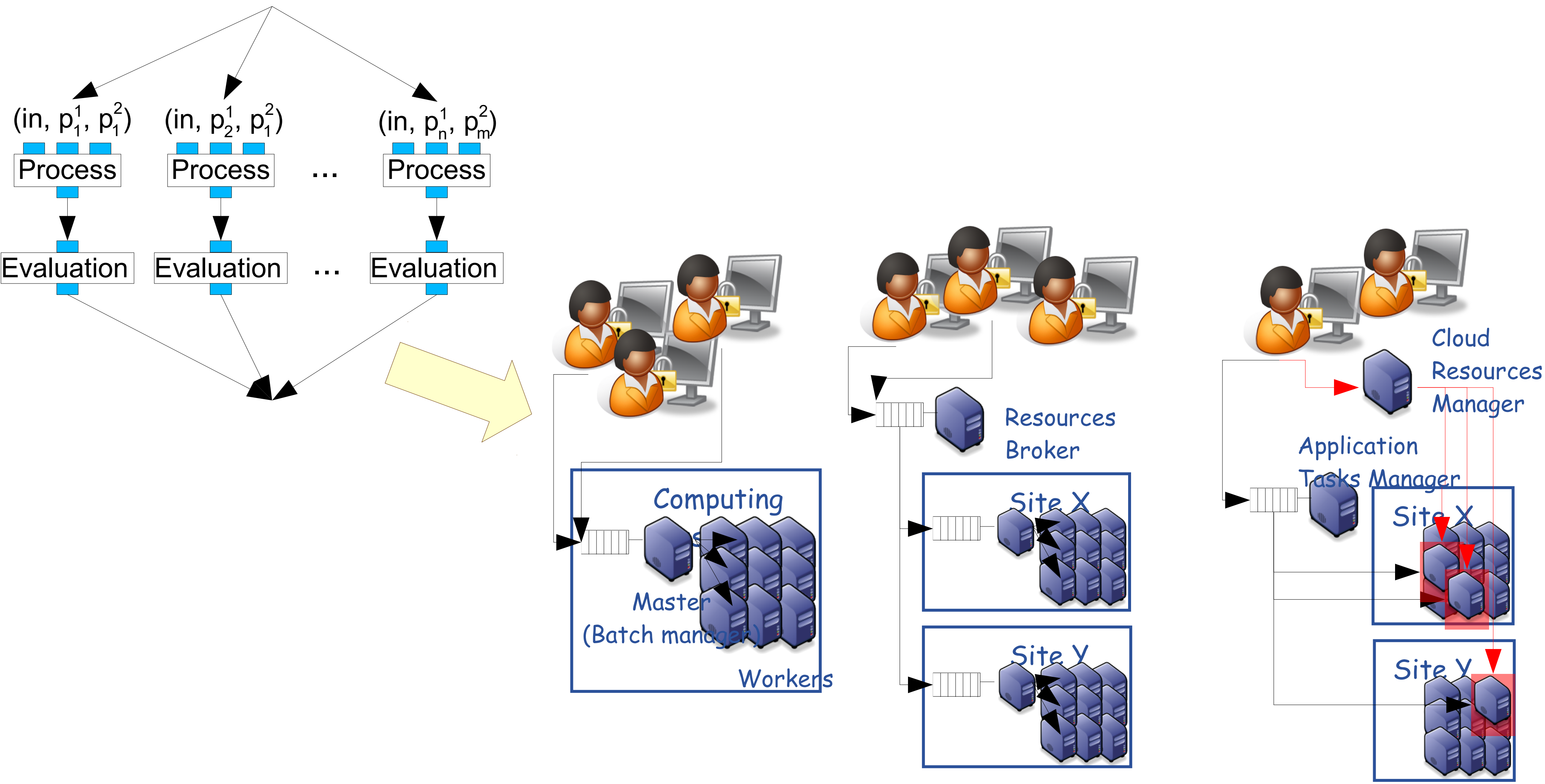
Low-level (system) view

- Abstract representation for the computational process
- Close to user problem modeling view
- Compact, humanly tractable

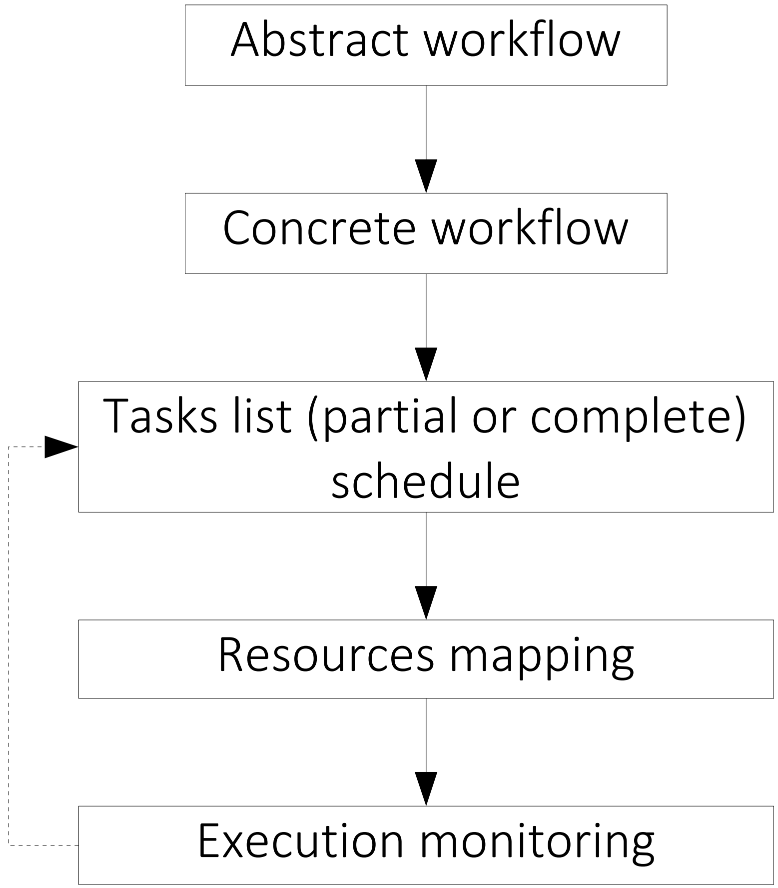
- Concrete (executable) representation
- Close to system description needs
- Detailed

Runtime

- Scheduling and mapping of workflow on computational resources



Workflow manager



- Different representations lead to different scheduling and mapping requirements

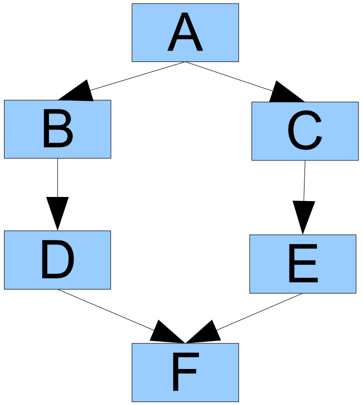
Workflow representations

Definitions of workflows

- Workflow Management Coalition:
  - "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"
- A workflow is a graph representing dependencies:
  - Data and control links
  - Control structures
- A workflow links the components of an application:
  - Services (Web-Services, DIET services, ...)
  - Tasks (JDLs)
  - Local codes
  - Human activities
- Workflows for distributed infrastructures
  - Human scale activities, legacy codes coupling

Illustration

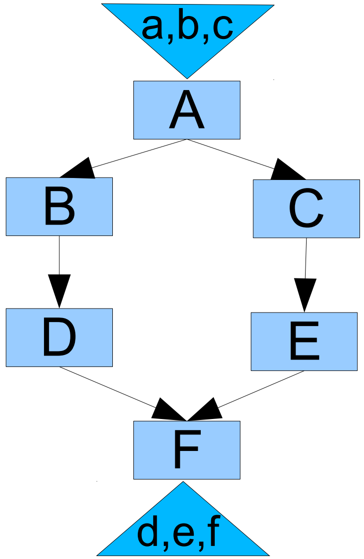
- Workflow



  - algorithm description

```
A
then ((B then D) and (C then E))
then F
```

- Data flow



  - Tasks computation

```
d = F(D(B(A(a))), E(C(A(a))))
e = F(D(B(A(b))), E(C(A(b))))
f = F(D(B(A(c))), E(C(A(c))))
```

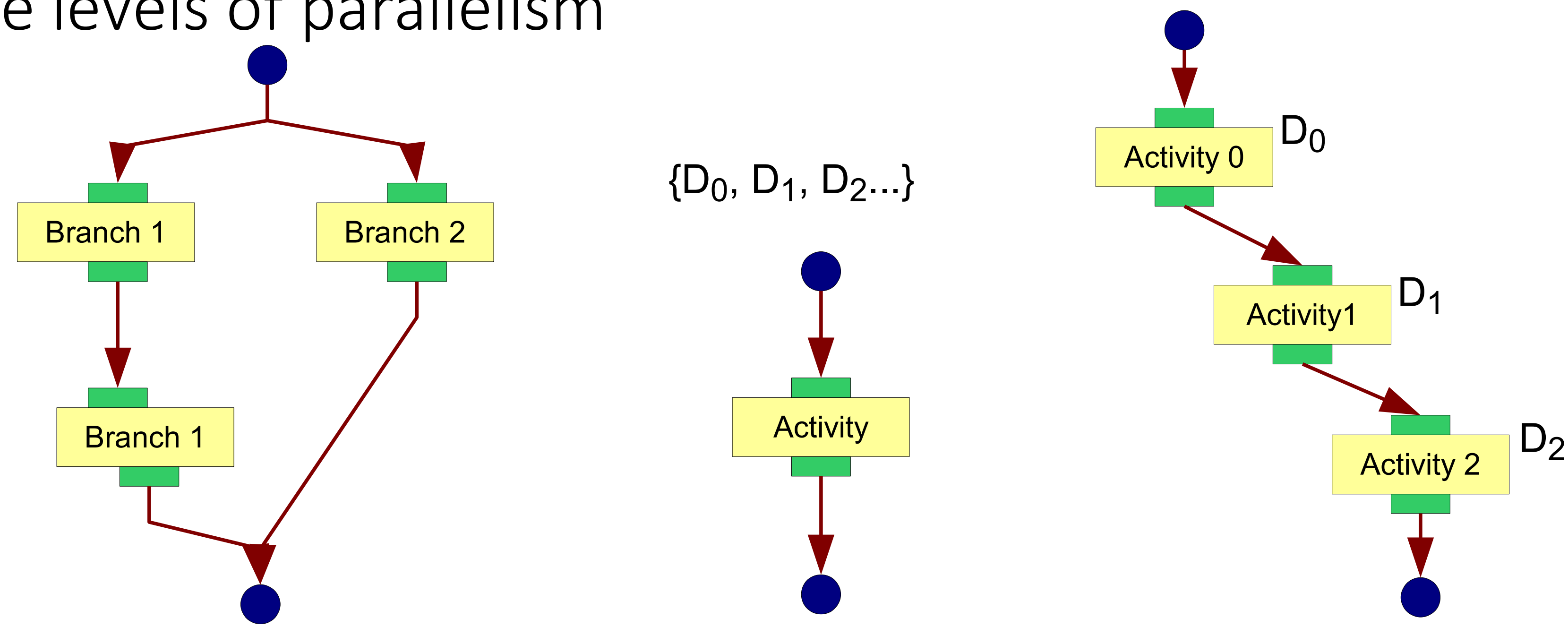
- Contains
  - Sequential (dependent) and parallel (independent) components
  - Implicit list of tasks to be computed
  - Command invocation details

Workflow representation

- Any programming language could be adopted...
  - From C to Makefiles
- ...but all do not fit well a distributed computing environment
  - Distributed computing workflows target human-scale activities
  - Prototyping is the rule
  - Scientific applications with heavy user communities and large data sets require high performance and/or high throughput
- Workflow languages for distributed systems
  - Coarse-grained
  - Data intensive
  - Heavy legacy code
  - Interfaced to external Job / Workload manager(s)
- Language simplicity vs expressiveness tradeoff
  - Separate design and enactment phase

Expression of parallelism

- Three levels of parallelism



Workflow parallelism (implicit in workflow graph) | Data parallelism | Services parallelism (pipelining)

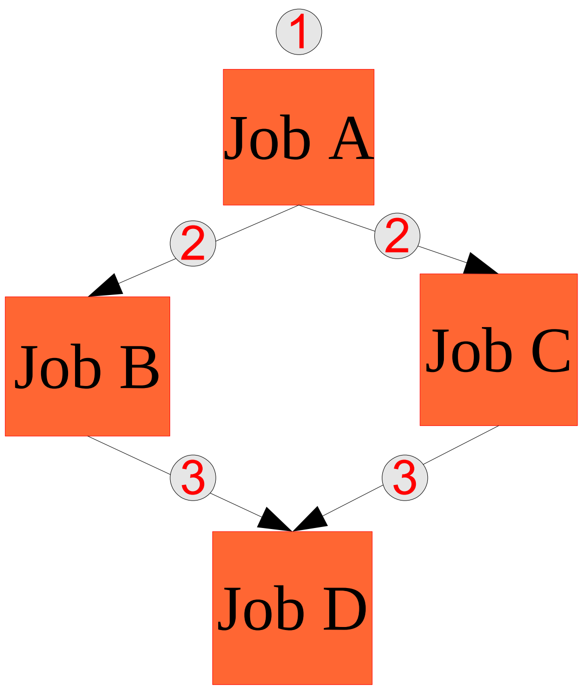
  - The expression of data and service parallelisms depend on the workflow language considered
  - The workflow engine has to be multi-threaded (or to perform asynchronous calls)
  - Data segments are puzzled

Workflow representation languages

- Directed task graphs
  - Simple dependency graphs
  - No complex structure
- Structured languages
  - Graphs with control structures
  - Scripts
- Data-driven workflows
  - Data exchanges are implicit dependencies
  - Data-driven computing model

Directed task graphs

- A task graph defines precedence constraints in a task set
- A task graph is a Directed Acyclic Graph (DAG)
- <u>Ex:</u> Condor DAGMan (*http://www.cs.wisc.edu/condor/dagman*)



Job A A.condor 1

Job B B.condor

Job C C.condor

Job D D.condor

Script PRE A top_pre.csh

Script PRE B mid_pre.perl $JOB

Script POST B mid_post.perl $JOB $RETURN

Script PRE C mid_pre.perl $JOB

Script POST C mid_post.perl $JOB $RETURN

Script PRE D bot_pre.csh

PARENT A CHILD B C 2

PARENT B C CHILD D 3

Retry C 3

Structured workflow graphs

- Control-centric, aka Business Workflow
- Example: Business Process Execution Language (BPEL)
  - From IBM (WSFL) and Microsoft (XLANG)
  - Build on Web-Services standards
- Defining the behavior of a process with a formal description of the messages exchanged by the Web-Services
- Specifies the behavior of all “partners” independently from their implementation
- The resulting process is itself a Web-Service

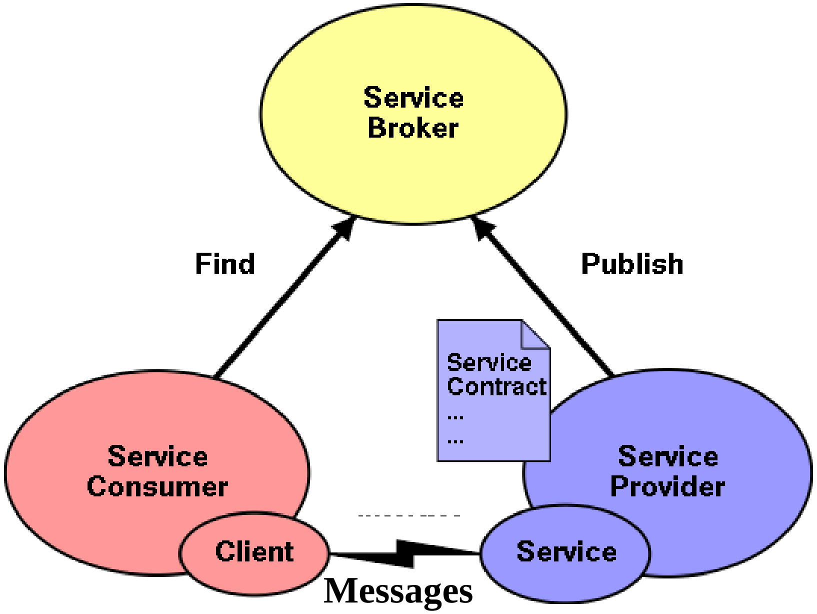
Software components and services

- Software components

  - Define and publish a standard interface
  - Interact by message exchanges
  - Ease (dynamic) composition
  - Modularity (no compilation time dependency)
  - High level message exchange protocols (format, types…)
- Services
  - Independent, self-sufficient software components
  - Can be invoked remotely
  - Can be dynamically created and destroyed

Service-Oriented-Architectures (SOA)

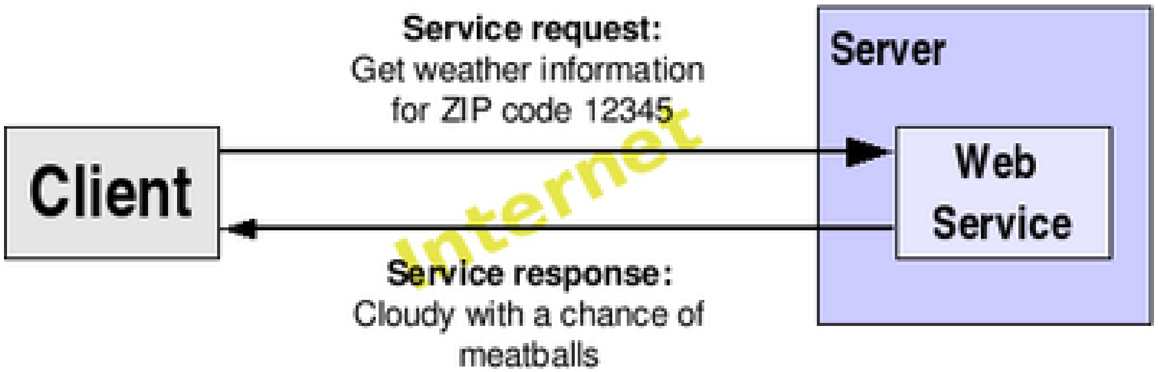
- Basic principles



- A service is an exposed piece of functionality with 3 properties:
  (1) The interface contract to the service is **platform-independent**
  (2) The service can be **dynamically located** and invoked
  (3) Services maintain a relationship that minimizes dependencies (**loosely coupling**)

Services in a SOA

- Client/Server interaction



- Platform and language independent
  - Client and server programs can be written in different languages, and run in different environments
- Self-describing
  - Only location is needed to invoke a service
  - Loosely coupled
- Based on the adoption of common standards

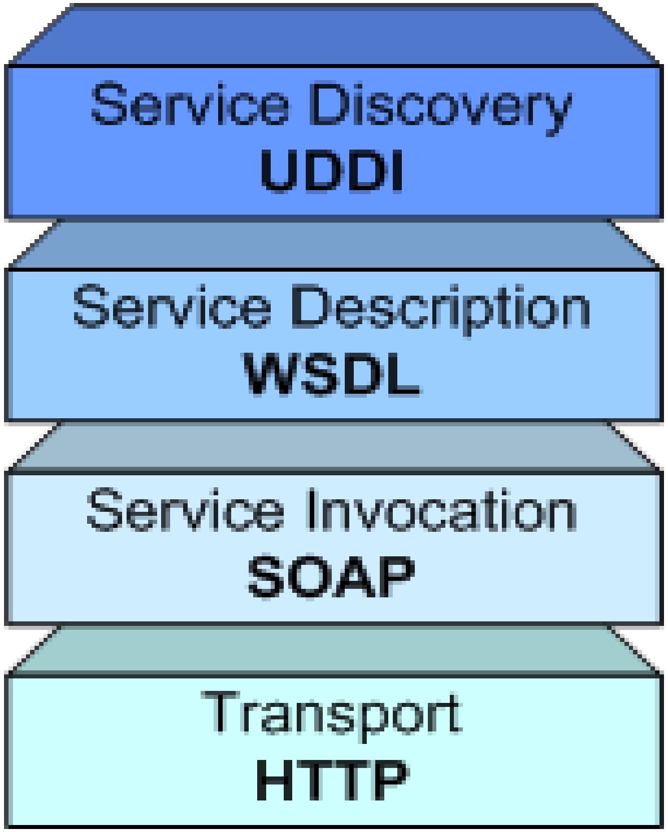
Web-Services standard

- Standardized by the W3C:
  - Contract / Interface format

    Web **S**ervices **D**escription **L**anguage
  - Messages format

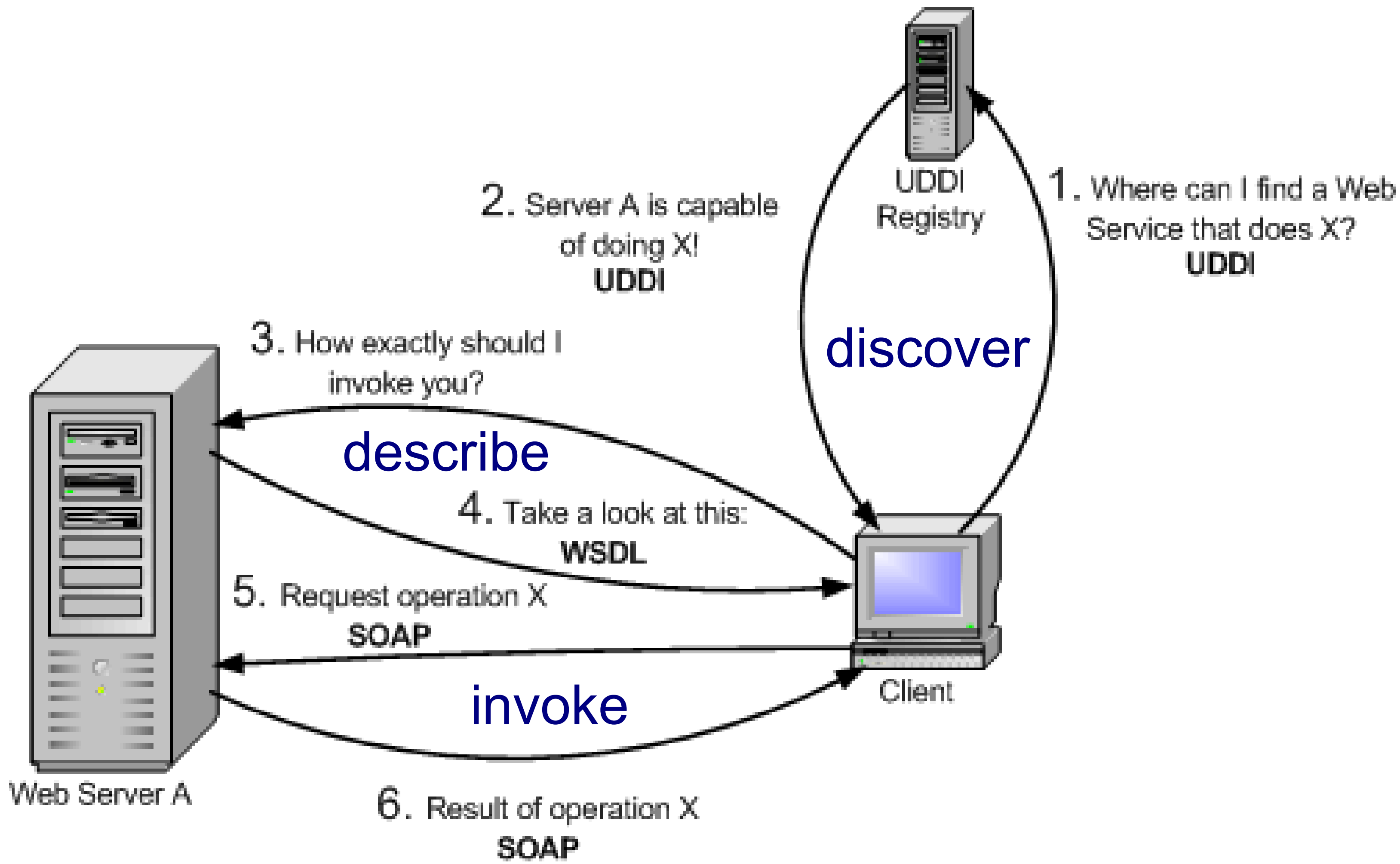
    Simple **O**bject **A**ccess **P**rotocol
  - Discovery format

    Universal **D**escription **D**iscovery & **I**ntegration
- Based on XML
  - Text format
  - Platform/language independence

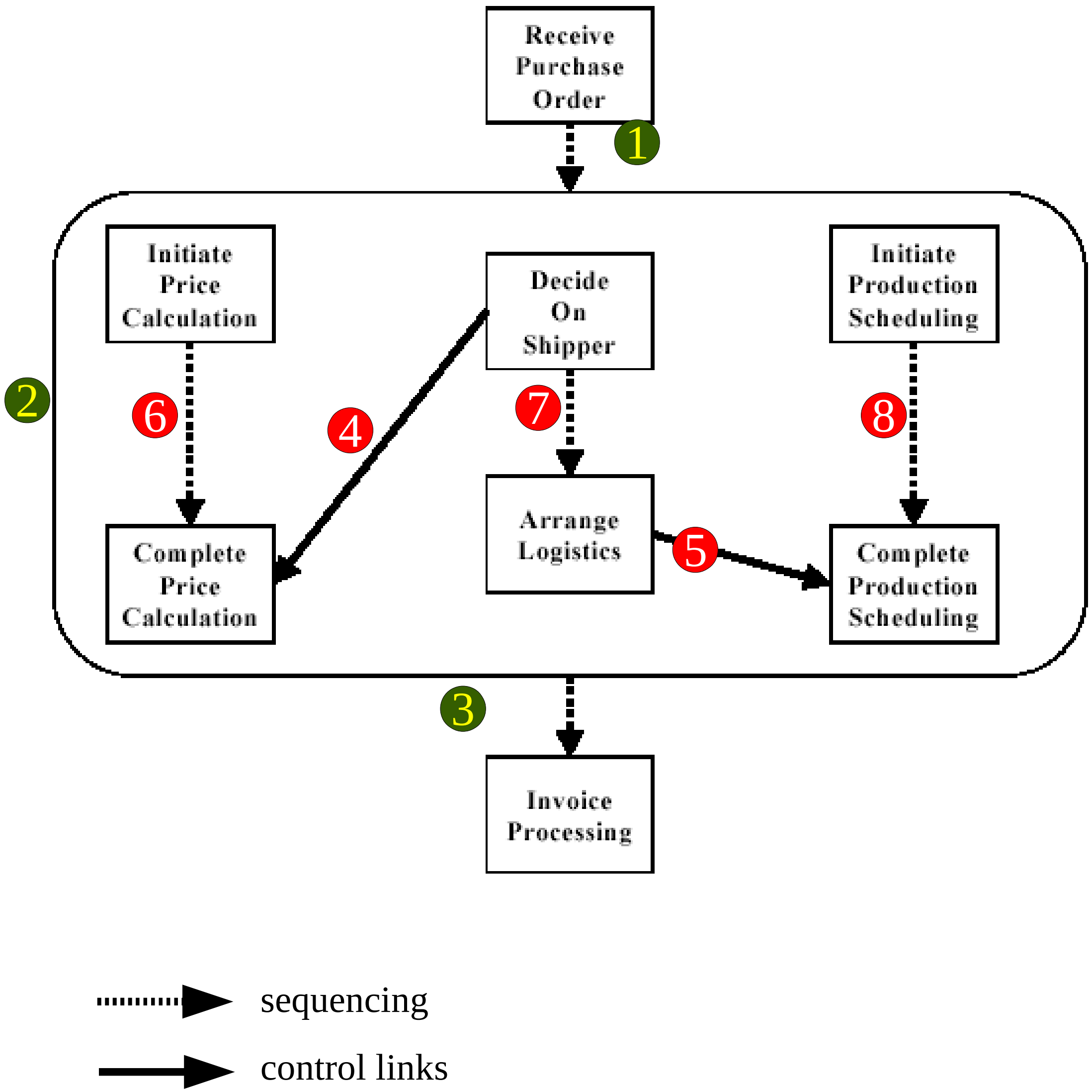


Web services

- Typical use case



BPEL web services workflow example



declarations

activities

```
<process name="purchaseOrderProcess">
    <partnerLinks>
    </partnerLinks>
    <variables>
    </variables>
    <faultHandlers>
    </faultHandlers>
    <sequence>
        <receive>        1
        </receive>
        <flow>
            <links>      4  5
            </links>
            <sequence>   6
            </sequence>
            <sequence>   7        2
            </sequence>
            <sequence>   8
            </sequence>
        </flow>
        <reply>          3
        </reply>
    </sequence>
</process>
```

BPEL: structured activities

- Can contain other activities:
  - <sequence>
    - *sequential execution*
  - <flow>
    - *parallel execution*
  - <pick>
    - *blocks the execution until a message/timeout occurs*
  - <switch>
    - *selects an activity according to a condition*
  - <while>
    - *iteration*
  - <scope>
    - *defines an activity with its own variables, handlers, ...*

BPEL: communicating activities

- Interact with the partners of the workflow:

  `<invoke>`

  sends a message to a port of a partner

  <receive>

  blocking wait of a message

  <reply>

  sends a message replying to a received message (by <receive>)

BPEL: other activities

- Other activities

  <assign>

  assigns a value to a variable

  <wait>

  waits for a given duration or until an instant

  <terminate>

  terminates the process

  <compensate>

  executes the compensate field (called by a fault handler)

  <throw>

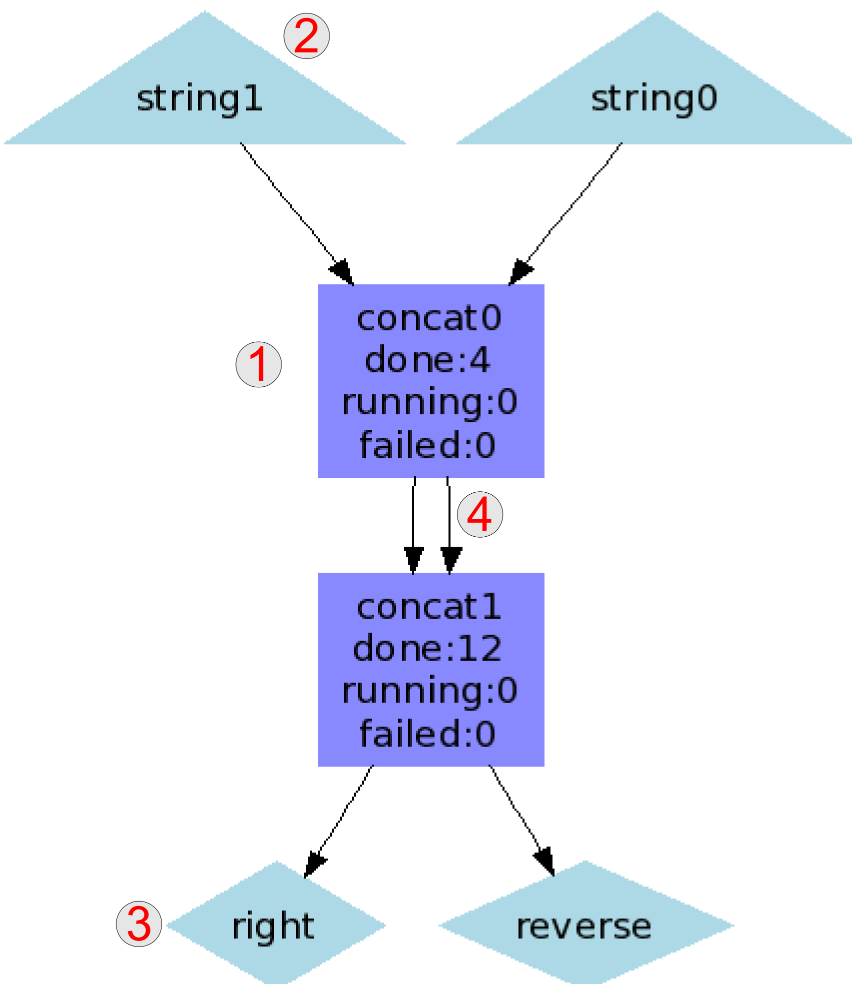
  throws an exception

  <empty>

  nop

Data-driven workflows

- Data-centric, aka scientific workflows
- Example: Simple Concept Unified Flow Language (Scufl)



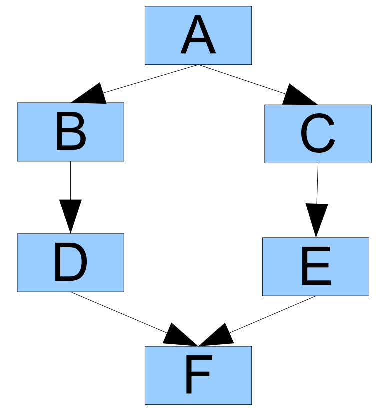

```
<processor name="concat0"> 1
  <arbitrarywsdl>
    <wsdl>http://colors.unice.fr/concat_service.wsdl</wsdl>
    <operation>concat</operation>
  </arbitrarywsdl>
  <iterationstrategy>
    <cross>
      <iterator name="string2" />
      <iterator name="string1" />
    </cross>
  </iterationstrategy>
</processor>
<processor name="concat1"> ...  </processor>
<source name="string1" /> 2
<sink name="right" /> 3
<link source="concat0:reverse" sink="concat1:string1" /> 4
```

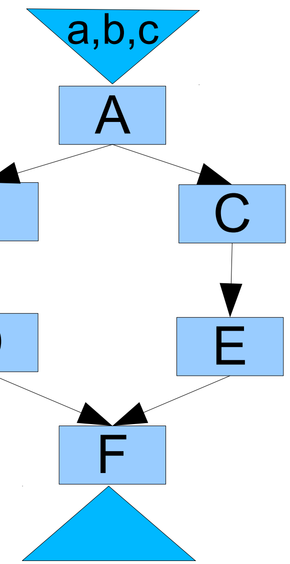
Taxonomy of workflow approaches

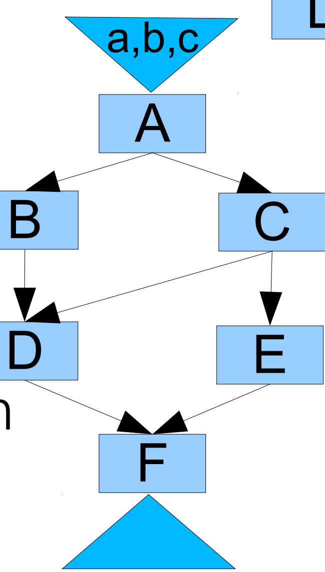
- Workflow descriptions
  - Business workflows languages (*e.g.* BPEL) → Control-centric
  - Scientific workflows languages (*e.g.* Scufl) → Data-centric
- Scientific workflows approaches
  - **Service-based** workflows (*e.g.* Scufl)
    - Independent expression of processings and input data
    - Dynamic description
    - Simple representations, complex optimization
  - **Task-based** workflows (*e.g.* DAGMan)
    - Tasks mixes data and processings
    - Static description
    - Complex representations, simple optimization

Scripting approach

- Any programming language may be use to describe a workflow
  - Workflow representation buried in code
- May use parallel control structures
  - Explicit description of parallel tasks
    - `Dopar(Activity(D0), Activity(D1))`
  - Parallel loops
    - ```
      D = { D0, D1, D2 }
      ParallelForeach(d in D) {
        Activity(d)
      }
      ```
  - Parallel threads
    - `Fork() / Join()`
- "1D" (linear) code representation
  - Opposed to 2D graphs
  - Parallelism implicitly expressed in 2nd dimension





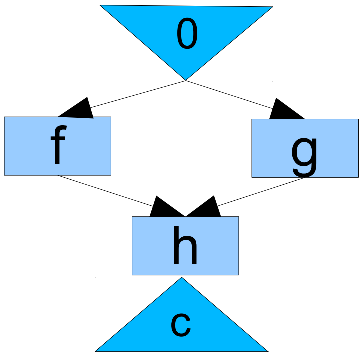

      ```

Future variables

- Traditional languages have blocking assignment instructions
  - ```
    a = f(0);    // variable assignment is blocking:
    b = g(0);    // execution of f() completes and a is
                 // assigned before g() is executed
    c = h(a, b); // h executed once a and b values are known
    ```
- Future variables are non-blocking assignment variables
  - Value read is blocking though
    - To preserve computations coherency
  - ```
    future a, b;
    a = f(0);    // variable assignment is non-blocking:
                 // f(0) is evaluated asynchronously
    b = g(0);    // g(0) is evaluated immediately
    c = h(a, b); // variable read is blocking
                 // a and b are evaluated before h(a, b)
    ```

Future variables

- Equivalent representations
  - ```
    Dopar { a = f(0), b = g(0) };
    c = h(a, b);
    ```
  - ```
    f1 = Fork(a = f(0));
    f2 = Fork(b = g(0));
    join(main, f1, f2);
    c = h(a, b);
    ```
  - Graph-based



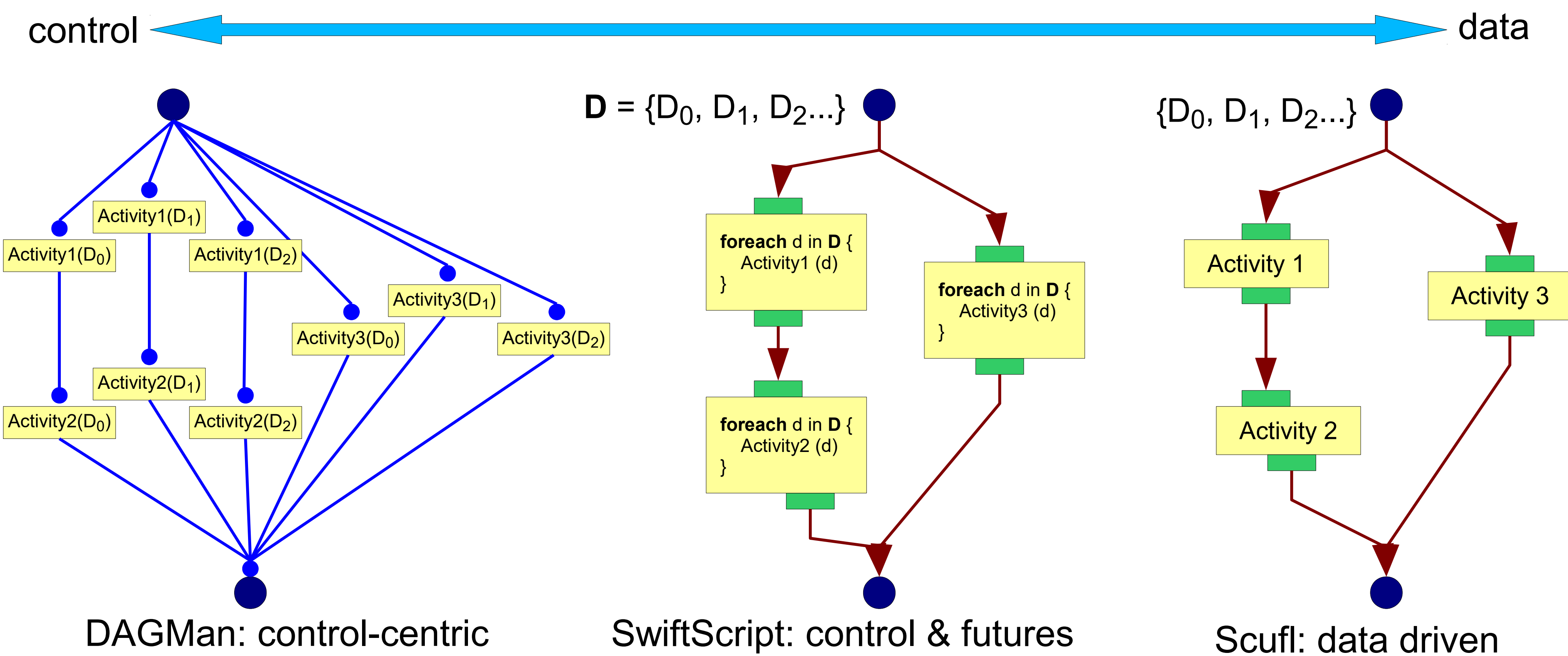
- Future variables introduce asynchronism in synchronous languages
  - Implicit representation of parallelism
  - e.g. SwiftScript workflow language

Parallel languages vs data-driven flows

- Parallel languages
  - Bounded: `foreach d in {D0, D1, D2}...`
  - Unbounded: `foreach d in D...`
  - Involves data synchronization at each data parallel service (no pipelining)
- Data-driven flows
  - Independent definition of data flows and data sets
  - Enable completely asynchronous enactment (data parallelism + pipelining): no data synchronization
  - May require explicit data synchronization barriers when this is needed
- Futures
  - Non-blocking assignment operations, blocking read
  - Data-centric approach, asynchronous execution

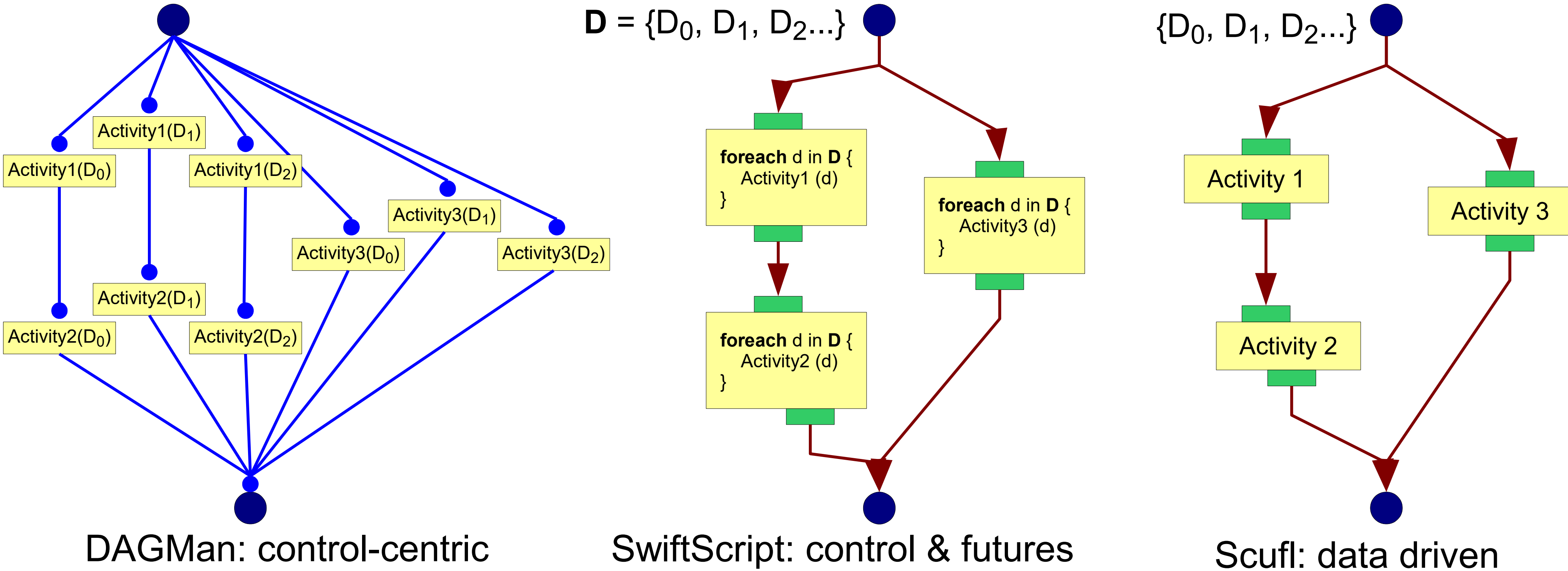
Control- vs data-driven language

- Different representations, similar semantics
  - Trade-off between workflow representation expressiveness and scheduling complexity



Representation of data parallelism

- Most languages express data parallelism



- DAGs: implicit in the (large-scale) workflow graph
- Parallel languages: explicit control structures
- Data-driven: transparent

Data-driven fits data-intensive well

- Main source of parallelism comes from data sets
  - Embarrassingly parallel
  - Parameter sweep
  - Single Process, Multiple Data (SPMD)
    - e.g. Map-Reduce
- Clear separation of application logic and experiment
  - Experiment = raw data and parameter data
  - This is a clear difference with BPEL (orchestration of Web Services) or other imperative programming inspired approaches
- Implicit description of parallelism

Arrays, activities and ports

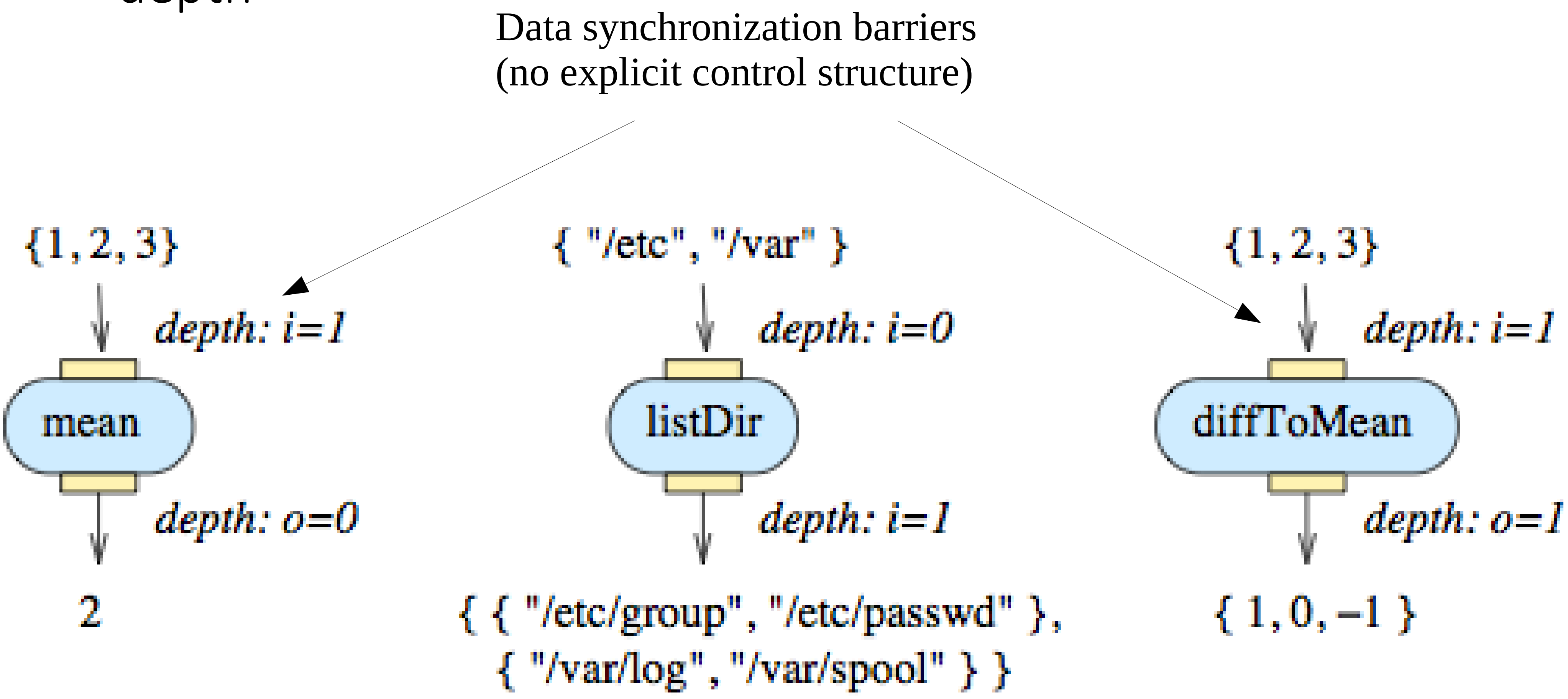
- Array programming principles
  - Lightweight syntax for handling arrays:

```
X+Y ≡ foreach i in indices(X) do
          Xi + Yi
      done
```

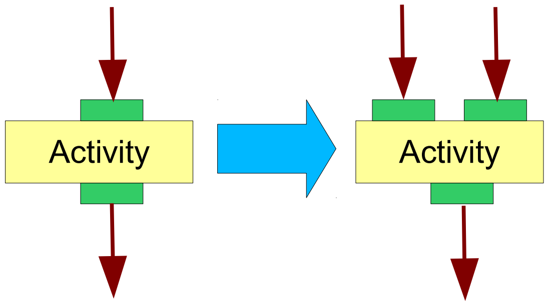
  - Convenient representation for vectorial processors
  - Extendable to any operation on arrays of values
  - Use nested arrays `X={{1, 2},{-1,-2}}` is a 2-nesting levels array
- Activities with array parameters
  - `A(X) ≡ { A(X0), ..., A(Xn) }`
  - `A{{1, 2},{-1,-2}} ≡ { {A(1), A(2)}, {A(-1), A(-2)} }`
- Ports depth
  - Activity ports have a defined depth that corresponds to the number of array nesting levels being synchronized:
  - `Mean({1, 2, 3}) = 2` if `Mean` has input port depth 1

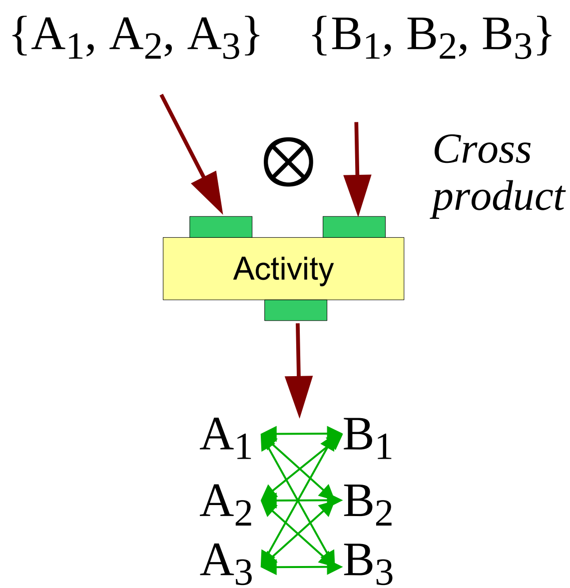
Nested arrays and ports depth

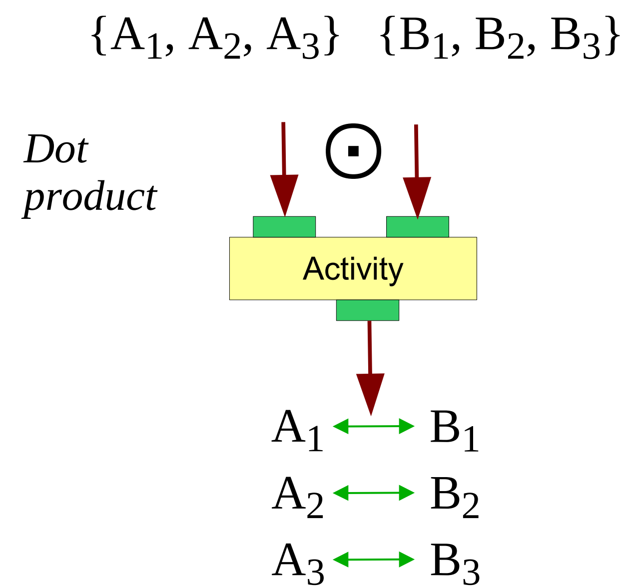
- Input arrays nesting levels are combined with activities ports depth



Iteration strategies







- In Scufl

$$\mathbf{A} \otimes \mathbf{B} \qquad \mathbf{A} \odot \mathbf{B}$$

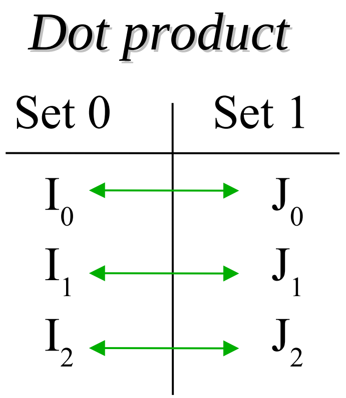
- Parallel language

```
foreach a in A
  foreach b in B
    fire Activity(a,b)
```

```
foreach i in indices(A)
  fire Activity(Ai,Bi)
```

Iteration strategies in a parallel WF

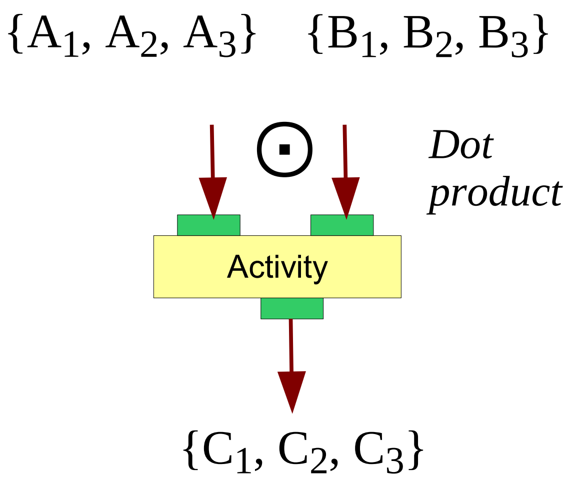
- Dot products assume ordered data sets

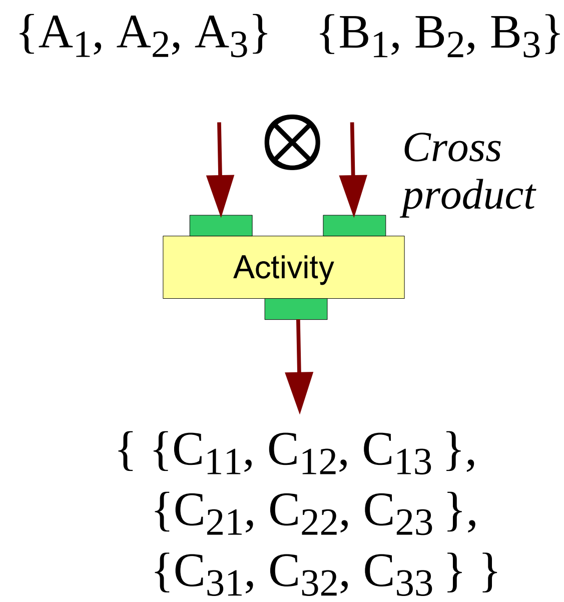


- No problem if:
  - Data parallelism is not present (order is preserved)
  - Service parallelism is not present (reordering is possible)
- Dot products in a data+service parallel execution:
  - Keep track of the data graph
  - Defines the dot product from the data graph

Iteration strategies and depths

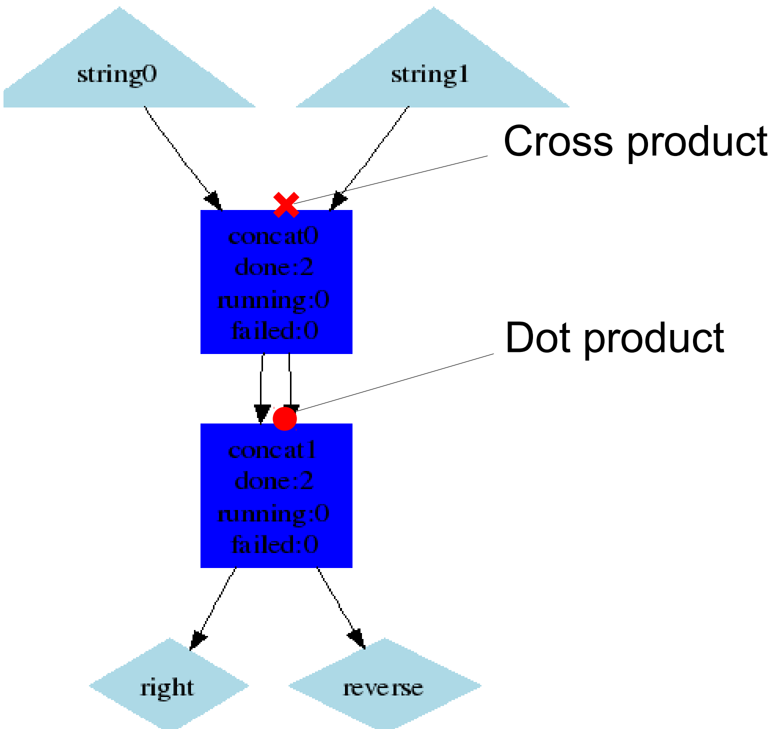
- Iterations strategies define output indexing strategies
- They may change data nesting levels



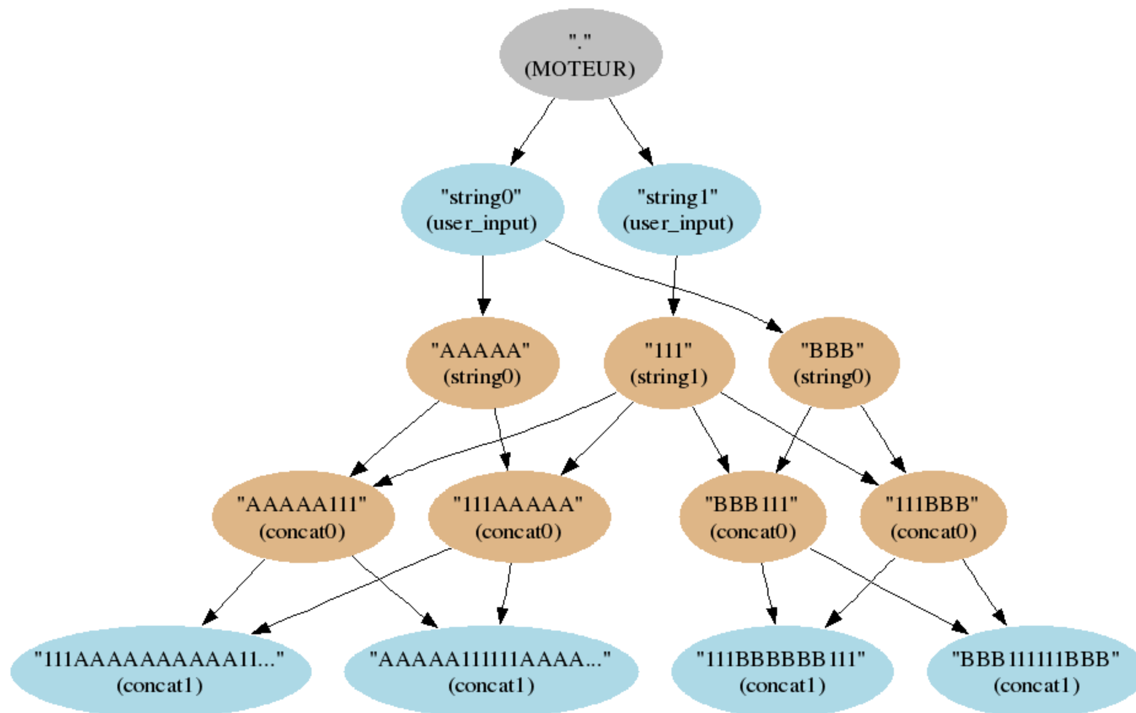


Data handling inside workflows

Services workflow



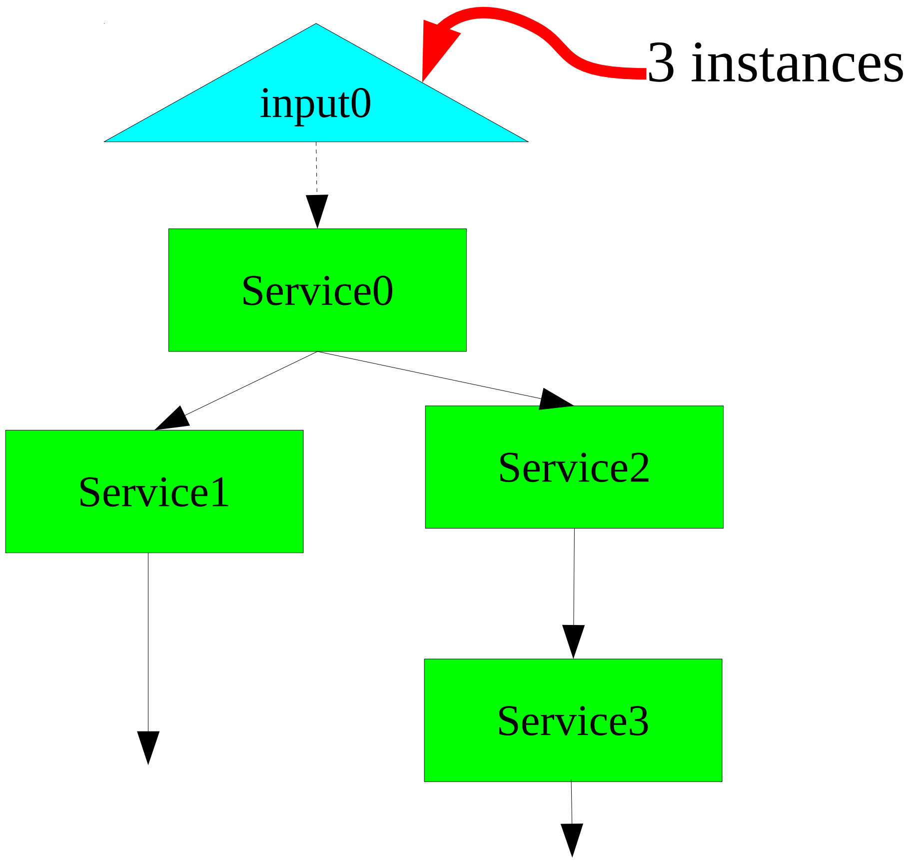
Data provenance graph



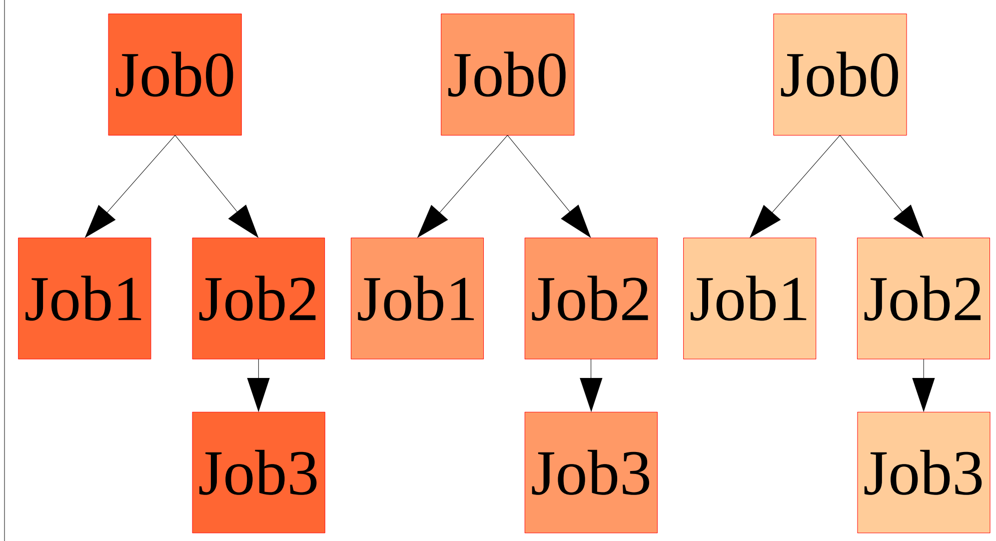
- This data representation allows to:
  - Handle dot products iteration strategies if data segments are puzzled
  - Retrieve results provenance

Data-driven workflows VS task graphs

Service-based workflow

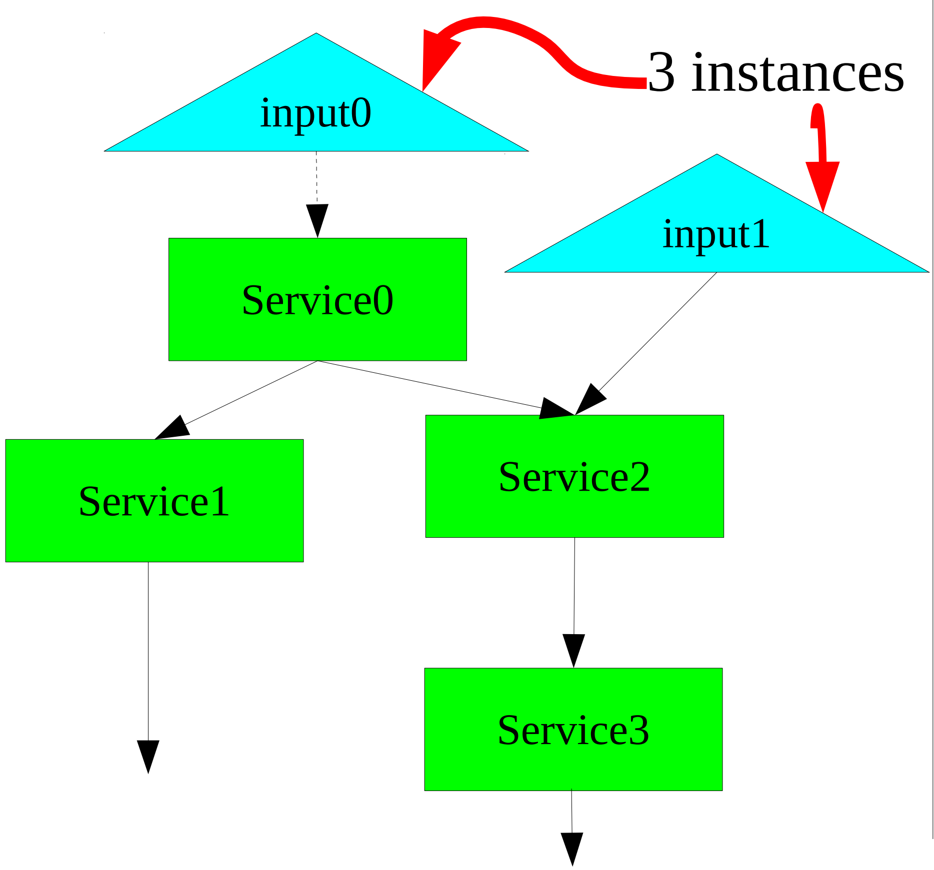


Task-graph

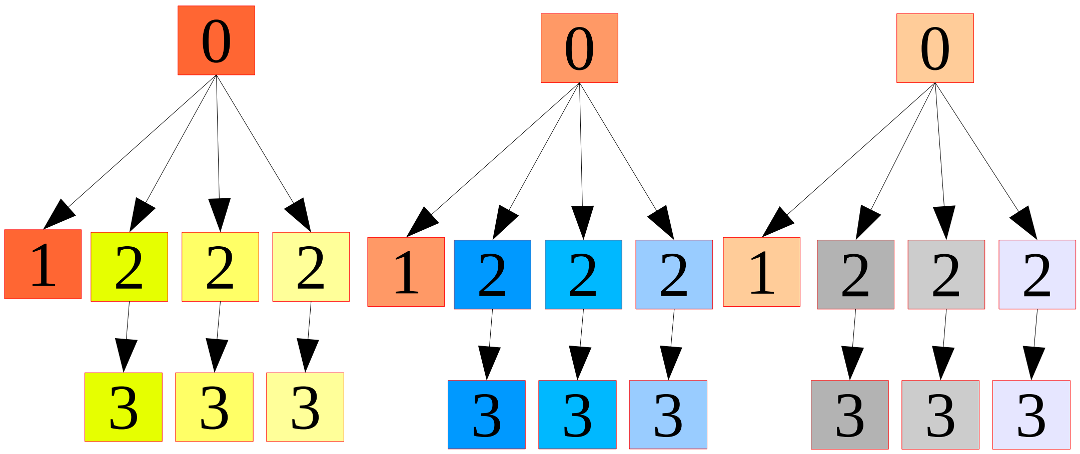


Data-driven workflows VS task graphs

Service-based workflow



Task-graph



- Data-driven workflows offer a data independent representation

Control structures or not?

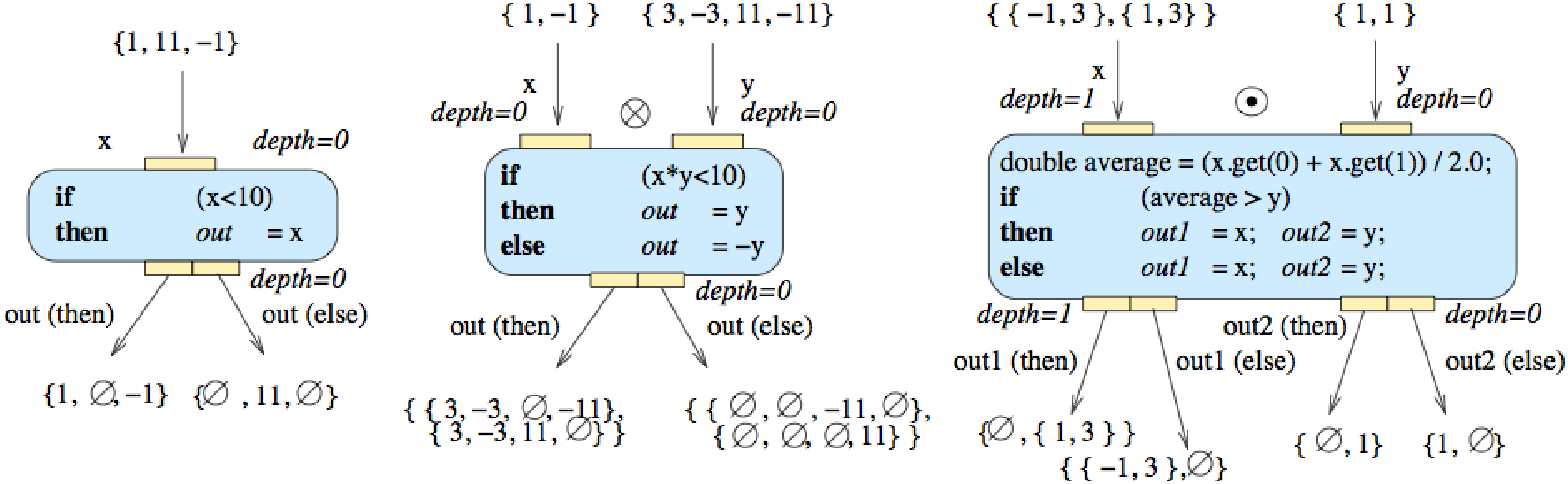
- Many data-driven / visual programming approaches do not define control structures
  - Hardly any, yet some (fail-if-false, fail-if-true)
- Directed Task Graphs do not include any loop
  - Because complex application logic is described inside the workflow activities
- Yet control on data flows is sometimes needed
  - Different execution conditions
  - Exceptions / Retry on errors at the application level
  - ...

Empty data element

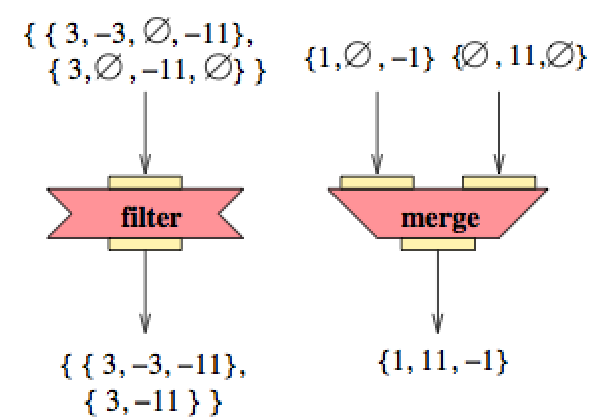
- Special *void* value $\emptyset$
- $\mathbf{A(\emptyset)} \equiv \emptyset$
  - No evaluation of the activity ($\emptyset$ is only known from the workflow engine)
- Iteration strategies semantics
  - $\mathbf{x} \odot \emptyset \equiv \emptyset \odot \mathbf{x} \equiv \emptyset$
  - $\mathbf{x} \otimes \emptyset \equiv \emptyset \otimes \mathbf{x} \equiv \emptyset$
- $\emptyset$ has an index in the array it belongs to

Conditionals

- Test expression variables mapped to input ports
- Dual “then / else” output ports
- Use of the special “void” data element to preserve indexing scheme coherence

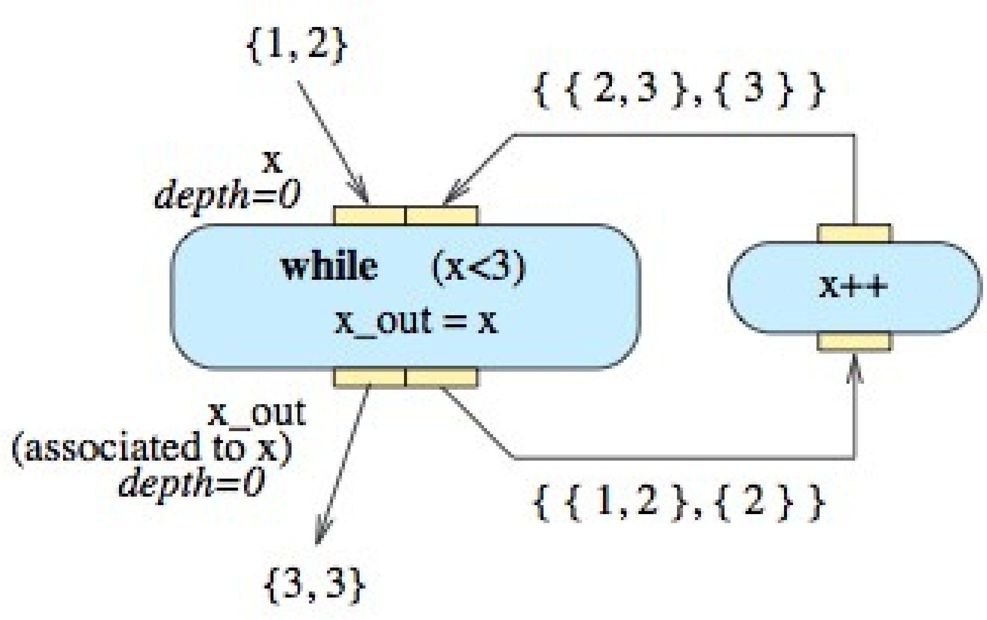


- Special filter / merge operations

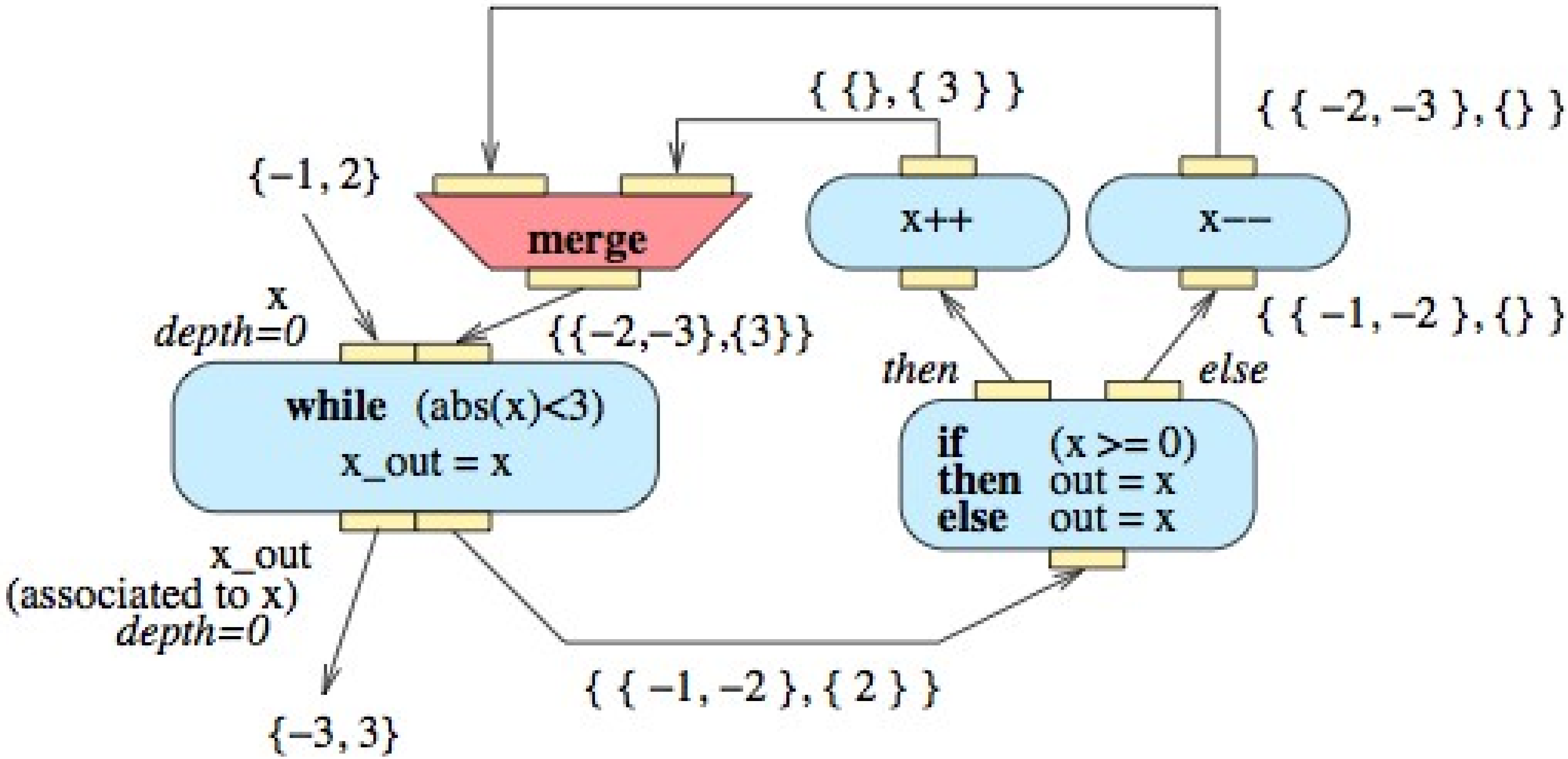


Loops

- Test expression variables mapped to input ports
- Dual “inner / outer” ports
  - Input ports: initialization / iterated values
  - Output ports: iterated values / end of loop



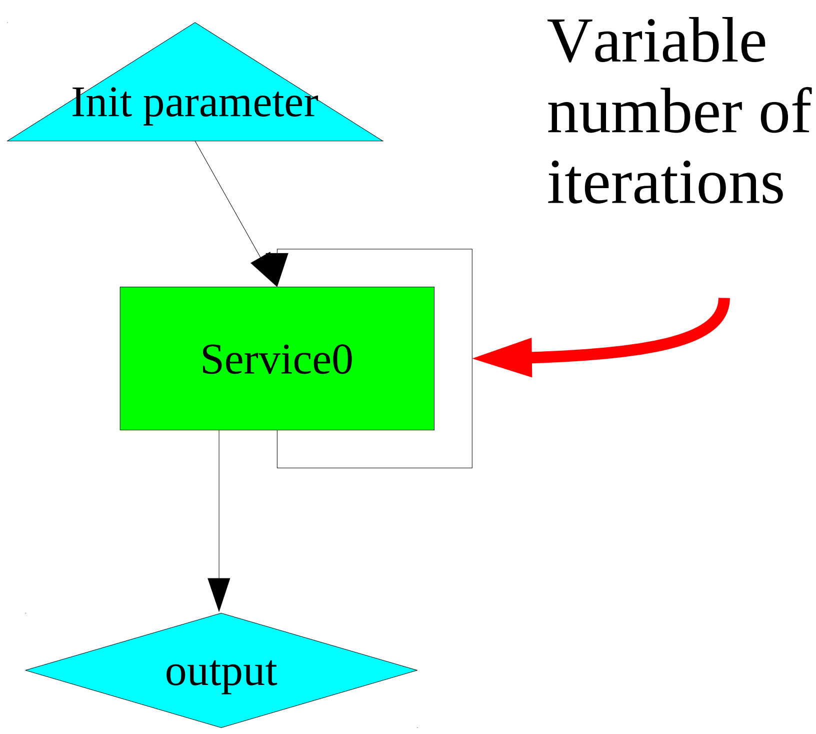
Example



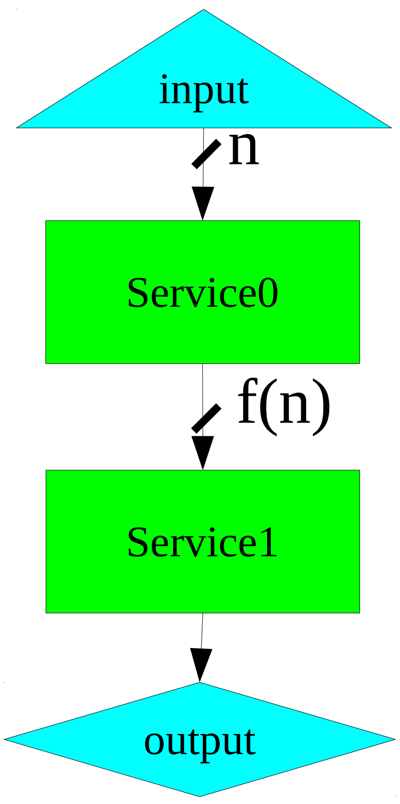
{ {}, { 3 } }
{ { −2, −3 }, {} }
{−1, 2}
merge
x++
x−−
x
depth=0
{{−2,−3},{3}}
{ { −1, −2 }, {} }
then
else
while (abs(x)<3)
x_out = x
if (x >= 0)
then out = x
else out = x
x_out
(associated to x)
depth=0
{ { −1, −2 }, { 2 } }
{−3, 3}

Data-driven workflows VS task graphs

- Dynamic data sets can be handled by service workflows:
  - Loops



Conditional data set size

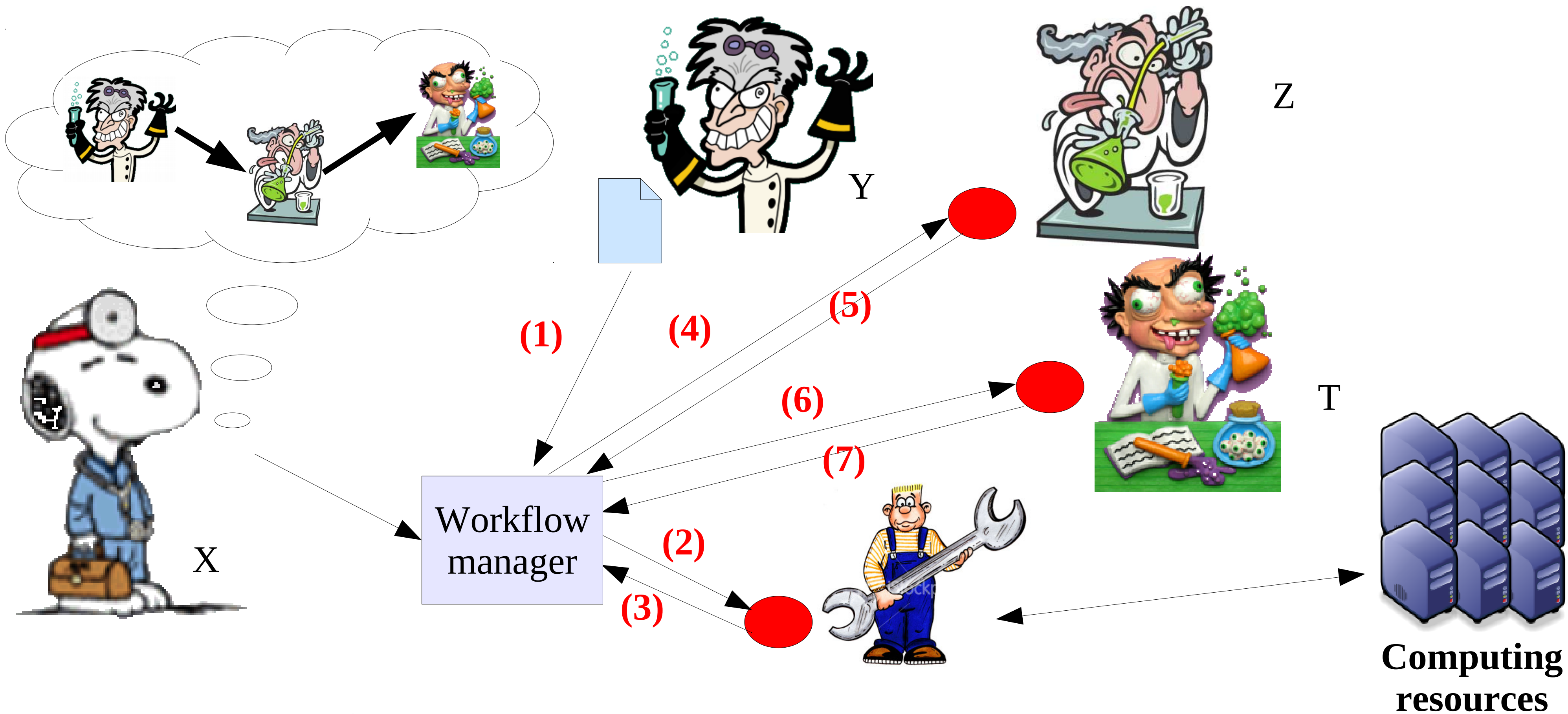


- Directed task graphs only allow static descriptions

Runtime

The workflow manager

- The workflow manager is a centralized engine that performs the calls to the services to execute the workflow
- It is a generic client to the services

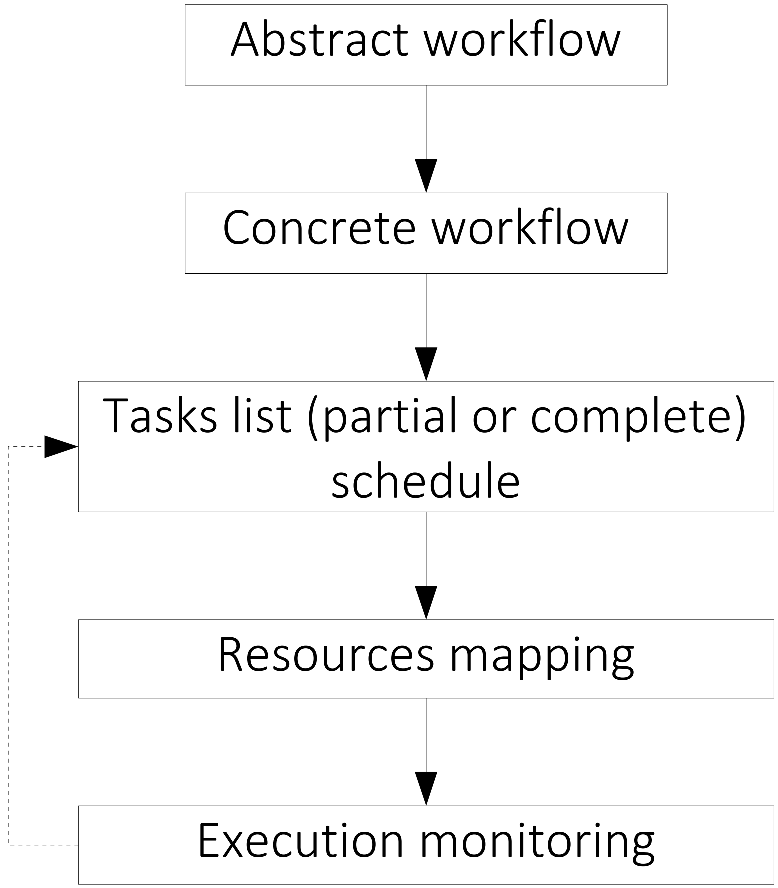


- Handling references to data is critical

Workflow managers

- Business workflows
    - ActiveBPEL engine, Apache ODE, ... (30 more)
    - JOpera *(http://www.jopera.ethz.ch)*
- Scientific workflows
    - Service Based
        - Taverna *(http://taverna.sourceforge.net)*
        - Triana *(http://www.trianacode.org/)*
        - Kepler *(http://kepler-project.org/)*
        - MOTEUR *(http://modalis.polytech.unice.fr/softwares/moteur)*
        - P-Grade portal *(http://portal.p-grade.hu)*
    - Task graphs
        - DAGMan (Pegasus and Chimera on top of it)
        - DIET MA-DAG

Workflow manager

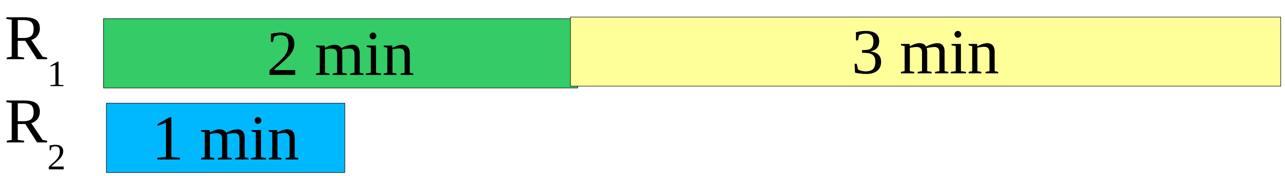


- Different representations lead to different scheduling and mapping requirements

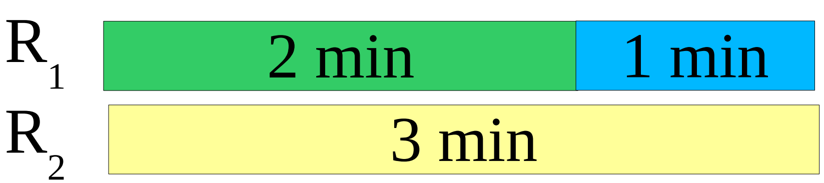
Scheduling

The scheduling problem

- Scheduling aims a finding a task execution and resources allocation planning to optimize one criterion or more.
- Example:
  - 3 tasks: 1 min | 2 min | 3 min
  - 2 resources: R_1 and R_2
  - Criterion: makespan (total execution time)
- Gantt charts of two schedules



➔ Makespan=5min



➔ Makespan=3min

Additional constraints

- Precedence constraints between tasks
  - Workflows/DAGs
- Communication costs
  - Data transfers between tasks
- Heterogeneous resources
  - Machines performances (CPU/memory)
  - Network bandwidth
- Dynamicity
  - Resources creation/deletion
  - Tasks creation/deletion

Scheduling task graphs

[Legrand, Robert 2003]

- Without communication costs:
  - The problem without resources limitations is polynomial
    - Scheduling ASAP (As Soon As Possible)
  - The problem with bounded resources is NP-hard

- With communication costs:
  - Both problems are NP-hard

- Heuristics are required to:
  - Set priorities to tasks
  - Allocate tasks to the processors

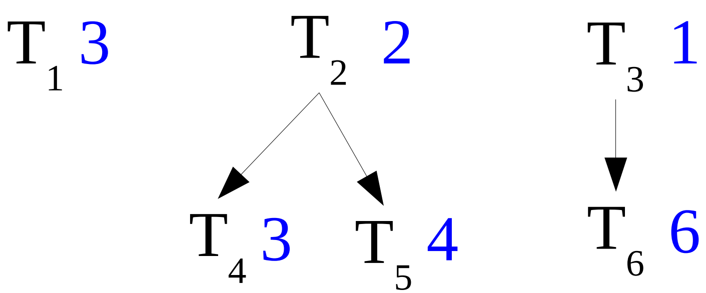
List heuristics for task graphs

- Idea: at each instant, allocate as many tasks as possible to the available processors (greedy algorithms)
- Defining priorities is required if there are more tasks than available processors
- Generic list scheduling algorithm
  - 1) Initialization
    - Compute the priority level of all the tasks
    - Set the priority queue as the list of free tasks (tasks without predecessors) sorted by decreasing priorities
  - 2) While it remains tasks to be executed
    - Add the new free tasks to the priority queue
    - If there are q available processors and r tasks in the queue:
      - Remove the min(q,r) first tasks of the queue and execute them

Tasks prioritization

- Prioritization based on the critical path:
  - Critical path of a task t: weight of the heaviest path starting from t
  - <u>Idea:</u> prioritize tasks with the heaviest critical path

- Example DAG:

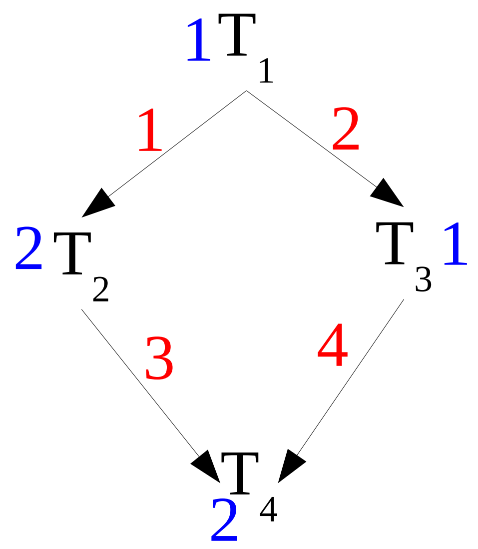


Tasks	T_1	T_2	T_3	T_4	T_5	T_6
Weights	3	2	1	3	4	6
Crit. path	3	6	7	3	4	6

Initial priority queue: (T_3,T_2,T_1)

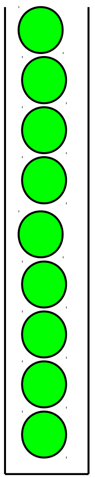
Communication costs

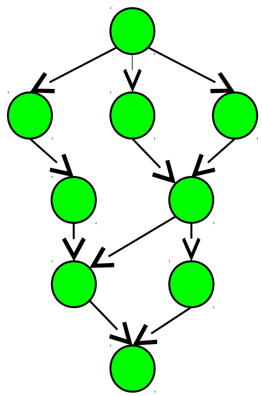
- DAG example:
  - Tasks weights in blue ; communication costs in red



  - Suppose as many processors as needed (4)
    - Sequential execution (no data transfer): 6 tops
    - Parallel execution: 10 tops (critical path: T1 → T3 → T4)
- A trade-off between, parallelization and communication costs has to be found

Heterogeneous resources





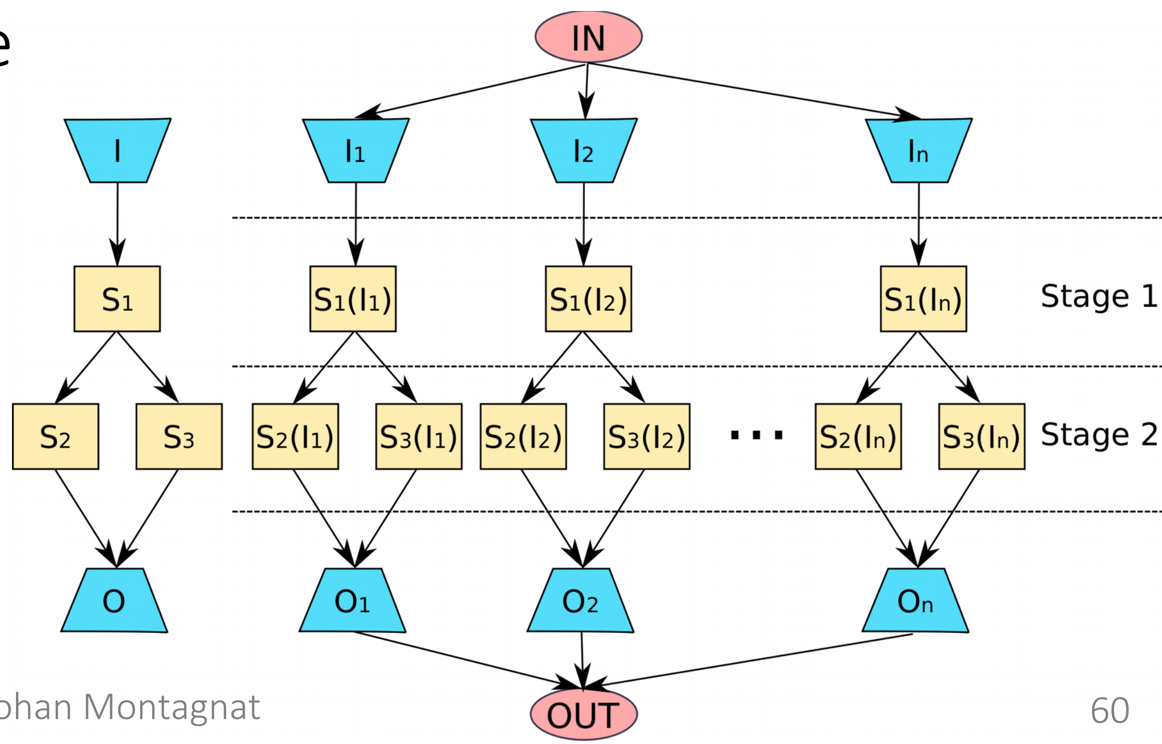
- Heterogeneous Earliest-Finish-Time (HEFT)
- List scheduling HEFT
  - Ordering
    - Set the weights of the tasks
    - Set the weights of the edges
    - Compute the rank (critical path) of each task.
    - Sort the tasks into a list L by non increasing order of their rank
  - Mapping
    - While the list L of tasks is not empty
      - Select the first task t of the list L
      - Select the resource r that has the earliest finish time for t
      - Allocate task t on resource r
      - Remove t from list L.

Scientific workflows scheduling

- Data intensive, scientific workflows
  - Each workflow activity (service) generates a large number of similar computation tasks: bags of tasks
  - Static workflow case: a critical path can be estimated
  - Dynamic workflow case:
    - Some non-deterministic activities (e.g. conditionals) break predictability
    - Sub-workflows without non-deterministic activities can be considered static
- On large batch distributed systems
  - Use statistics to estimate tasks execution and data transfer time
- On clouds
  - Estimate the number of resources to allocate
  - May vary during workflow execution time

Workflow-based resource planner for clouds or pilot jobs

- Executing application in several stages
- Hypothesis
  - Known duration of each task (low / predictable variability)
  - Resources are dedicated to the task (cloud computing)
  - No loop nor other unbounded control structures (deterministic)
- Optimizing number of computing resources and network bandwidth for each stage



Workflow cost computation model

- T_i: estimated execution time of stage i
- Computing resources

$$C_r = c_r \times \sum_{i=1}^{s} m_i \times (Td_i + T_i)$$

- Network bandwidth

$$C_b = c_b \times \sum_{i=1}^{s} (Td_i + T_i) \sum_{j=1}^{k_i} b_{i,j}$$

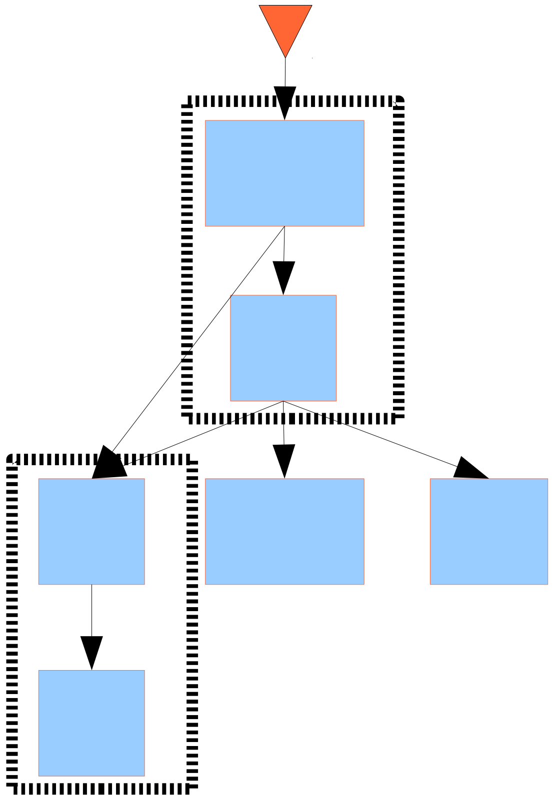
- Total execution cost

$$C = C_r + C_b$$

s	Number of execution stages
m_i	Number of nodes used in stage i ($m_i \leq m_{max}$)
Td_i	Deployment time of stage i
T_i	Estimated execution time of stage i (in second)
c_r	Per-second cost of a computing node
k_i	Number of links used in stage i
$b_{i,j}$	Bandwidth of link j used in stage i (in Mbps)
c_b	Per-Mbps cost of bandwidth

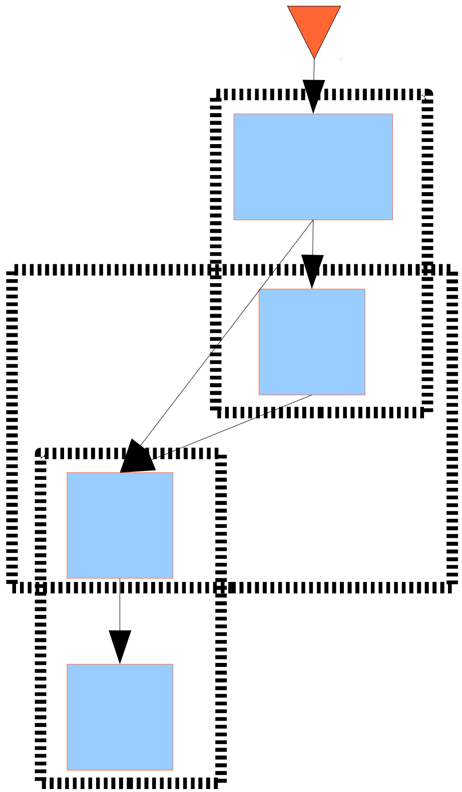
Grouping strategy

- To reduce the impact of jobs submission overhead
- Activities grouping
  - 6 services – 2 grouped pairs
  - 4 job submissions/input data set



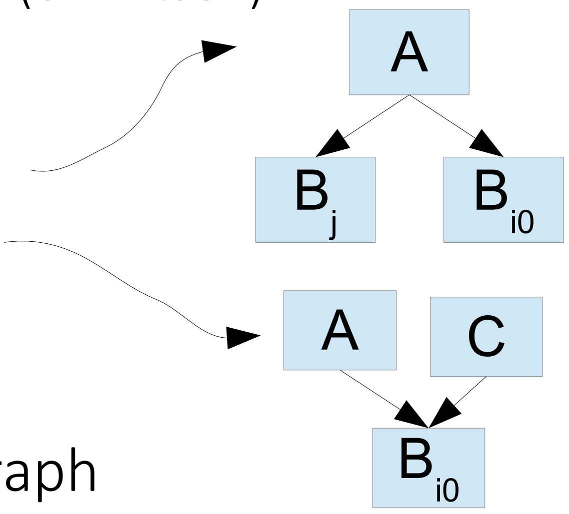
Recursive grouping

- 4 services – 3 grouped pairs
- 1 job submission/input data set



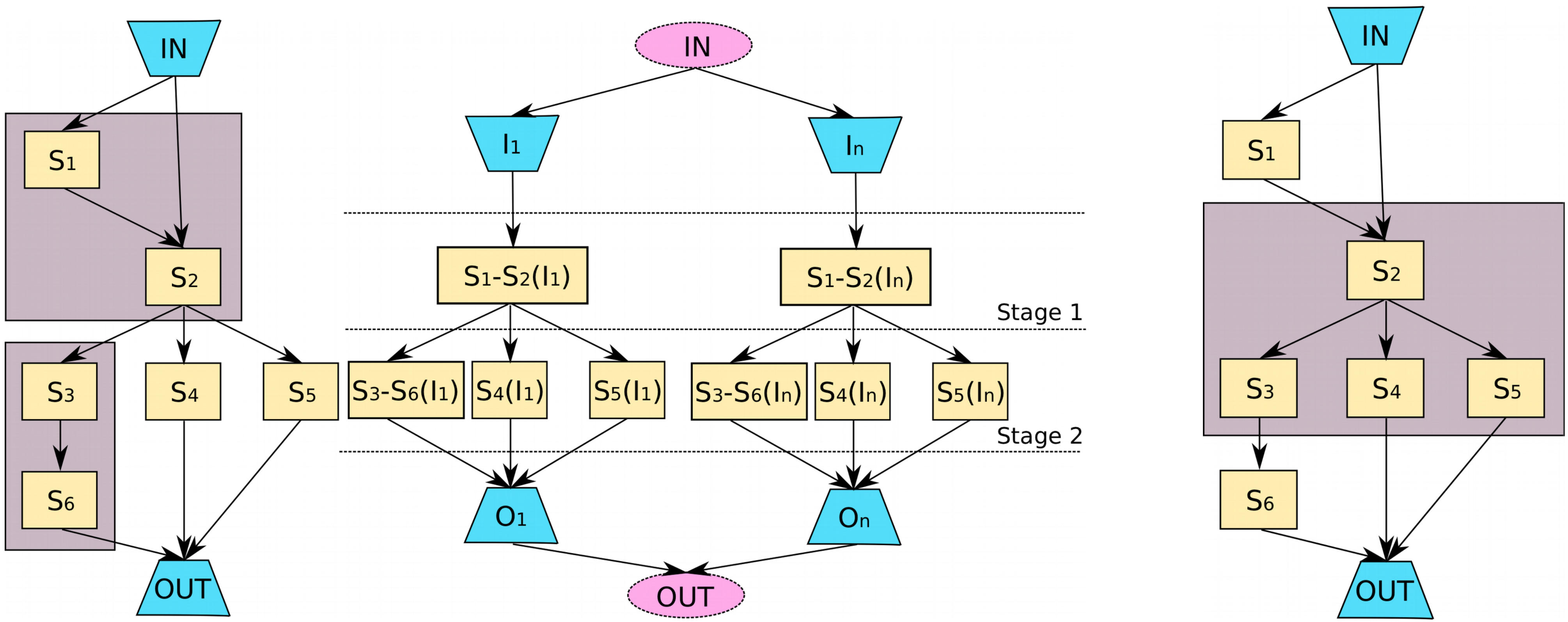
Grouping strategy

- Let A be a service of the workflow and $\{B_0,...B_n\}$ its children
- For grouping A and B_{i0}: no parallelism loss <=> (1) & (2)
  - (1) B_{i0} is an ancestor of every B_j
  - (2) Every ancestor of B_{i0} is an ancestor of A (or A itself)
- No parallelism loss => (1) & (2)
  - ¬(1) => parallelism between B_j and B_{i0} is broken
  - ¬(2) => parallelism between A and C is broken
- (1) & (2) => no parallelism loss
- This rule is recursively applied on the workflow graph



Grouping vs parallelism loss

- Aggressive grouping leads to parallelism loss
- Estimate whether grouping gain compensates for parallelism



Resources mapping

Condor matchmaker example

- Workload management
  - Heterogeneous resources
- Deliver High Throughput Computing
  - For many experimental sciences, the computing throughput matters. Focus is not instantaneous computing power but the amount of computing that can be harnessed over a long period.
  - HTC is a 24-7-365 activity: fault tolerance is critical
- Batch-oriented system
  - Batch extended with Job Control Languages to face grid heterogeneity
- Distributed computing IS difficult
  - team of ~35 faculty, full time staff and students (U. Winsconsin)
  - established in 1985
  - Faces software/middleware engineering challenges in a UNIX/Linux/Windows/OS X environment

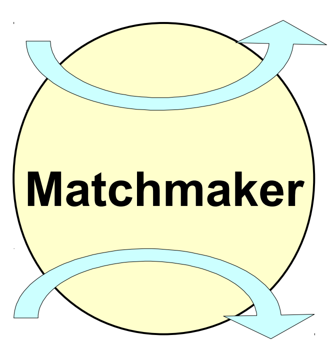
Matchmaking heterogeneous resources

- Run jobs in a variety of environments
  - Local dedicated clusters (machine rooms)
  - Local opportunistic (desktop) computers
  - Grid environments; Interface to other systems

- Matchmaking process

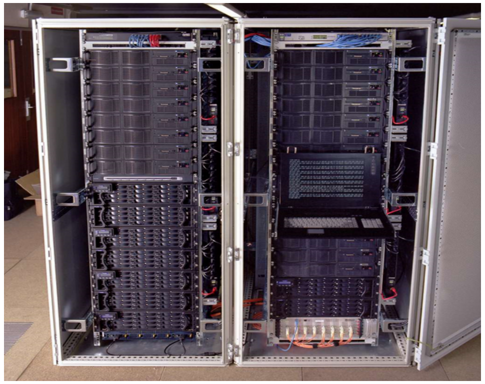
I need a mac for this code to run

I need a linux box with 2Gb RAM



Desktop Computers

Dedicated Clusters



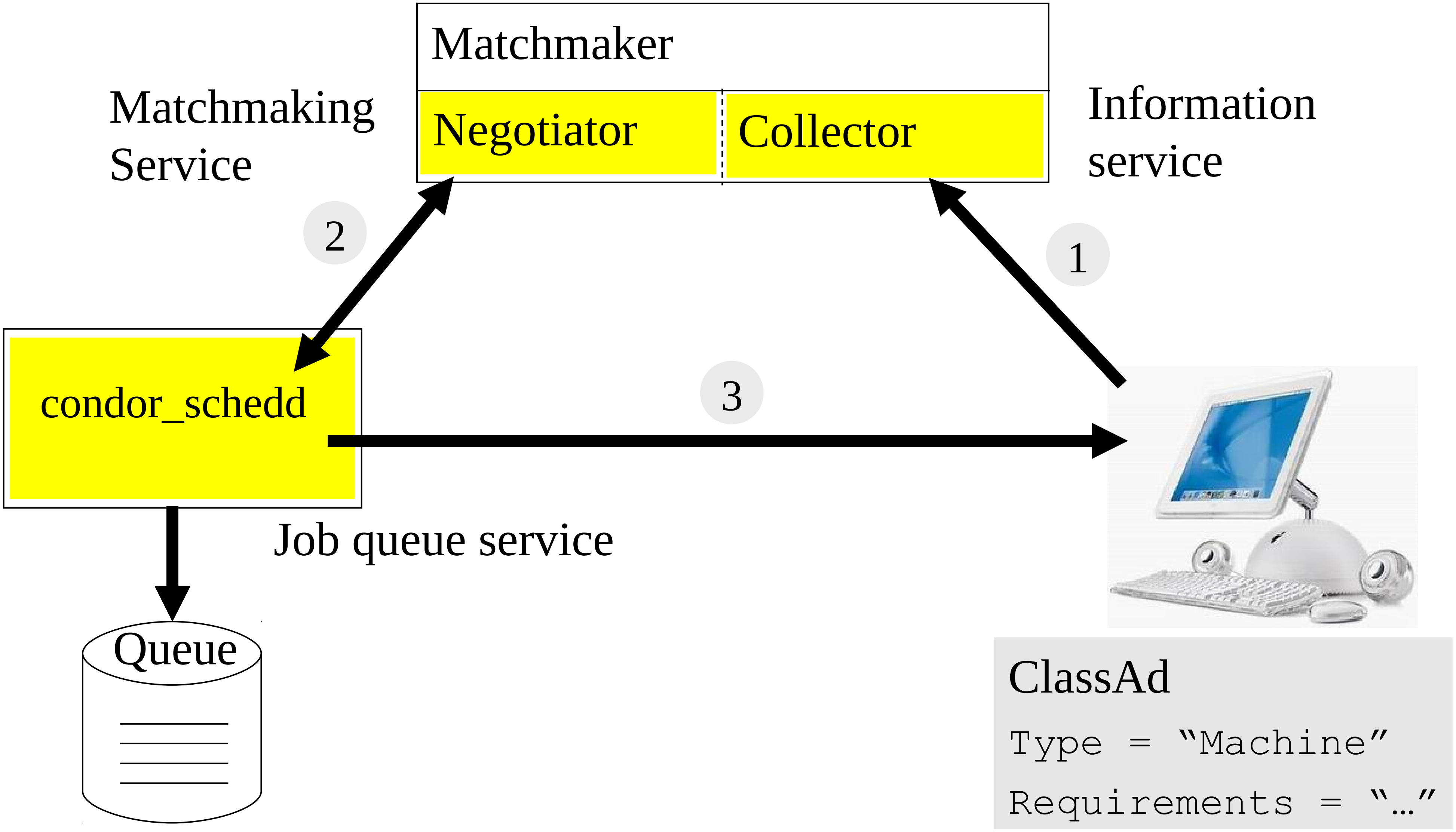
Matchmaking

- Condor conceptually divides people into three groups
  - Job submitters
    - I need Linux, and I prefer faster machines
  - Machine owners
    - I prefer jobs from the physics group
    - I will only run jobs between 8pm and 4am
    - I will only run certain types of jobs
    - Jobs can be preempted if something better comes along
  - Cluster administrator
    - When can jobs preempt other jobs?
    - Which users have higher priority?
    - Do some groups of users have allocations of computers?

Matchmaking

- Matchmaking is two-way
  - Job describes what it requires:
    - I need Linux && 8 GB of RAM
  - Machine describes what it provides:
    - I will only run jobs from the Physics department
- Matchmaking allows preferences
  - I **need** Linux, and I **prefer** machines with more memory but will run on any machine you provide me
- ClassAds Job Description Language (JDL)
  - Stating facts
    - Job's executable is analysis.exe
    - Machine's load average is 5.6
  - Stating preferences
    - I require a computer with Linux

Matchmaking diagram



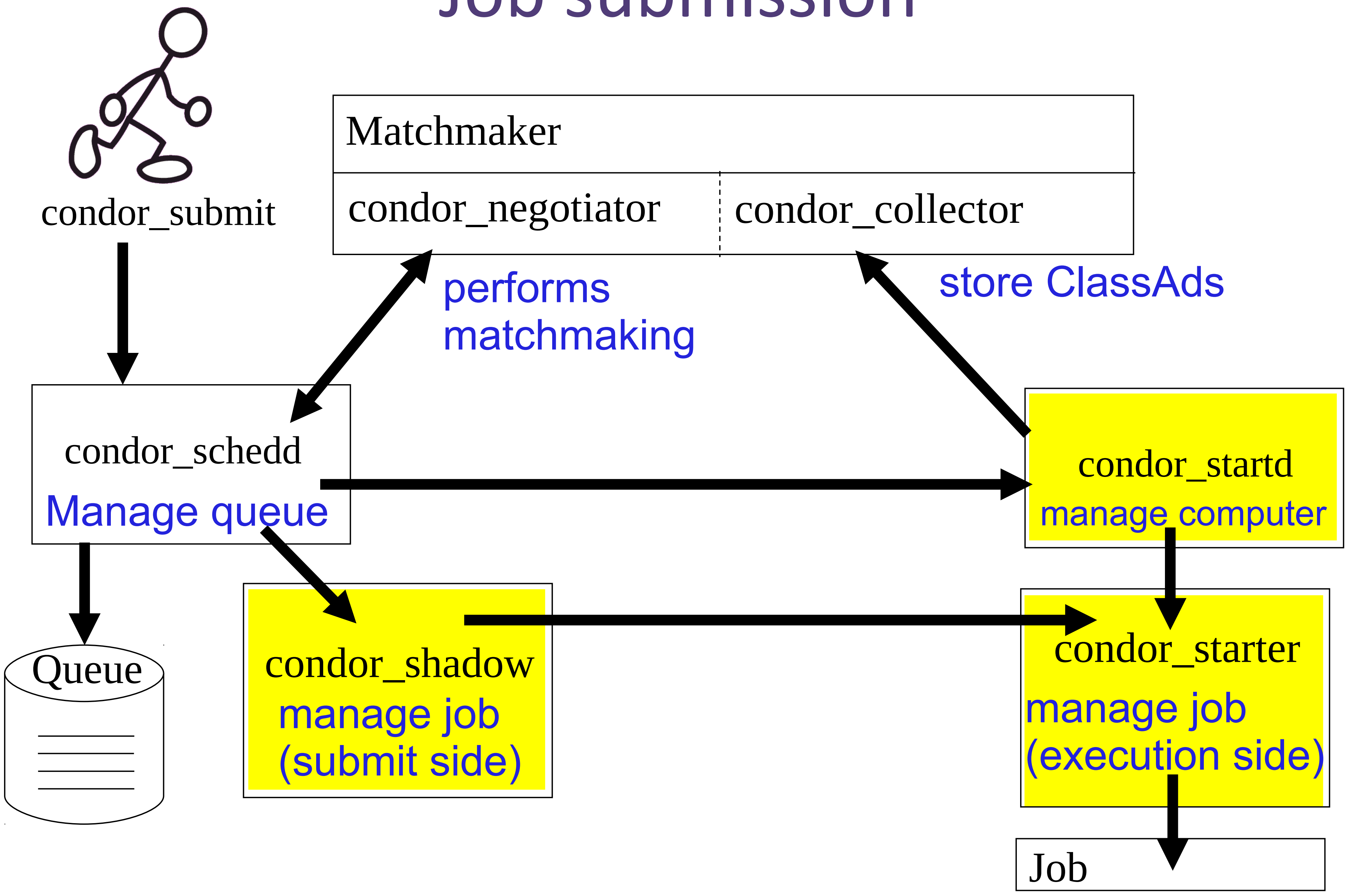
Including dynamic information (load...)

ClassAds JDL

- ClassAds are:
  - Semi-structured
  - Attribute = Expression
  - Schema-free, user-extensible
- Extensible declaration
  - `HasJava_1_4 = TRUE`
  - `ShoeLength = 7`
- Extensible matchmaking
  - `Requirements = OpSys == "LINUX" && HasJava_1_4 == TRUE`
- Example

```
MyType       = "Job"
TargetType   = "Machine"
ClusterId    = 1377
Owner        = "roy‟
Cmd          = “analysis.exe‟
Requirements =
    (Arch == "INTEL")
&& (OpSys == "LINUX")
&& (Disk >= DiskUsage)
&& ((Memory * 1024)>=ImageSize)
…
```

Job submission



Job submission

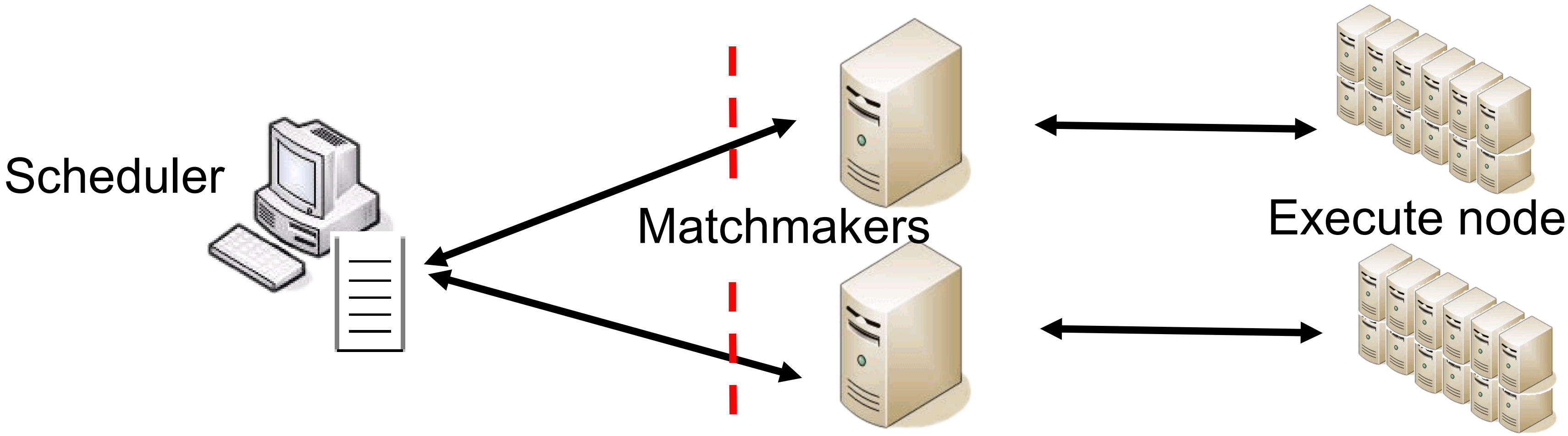
- Users submit jobs to scheduler
  - Jobs described as ClassAds
  - Each scheduler has a queue
  - Scheduler / queues are not centralized
- Negotiator
  - collects list of computers
  - contacts each schedd (What jobs do you have to run?)
  - compares each job to each computer to find a match
    - Evaluate requirements of job & machine in context of both ClassAds
    - If both evaluate to true, there is a match
- Fault tolerance scheduler
  - Resubmission
  - Fail-over scheduler

Large-scale computing techniques

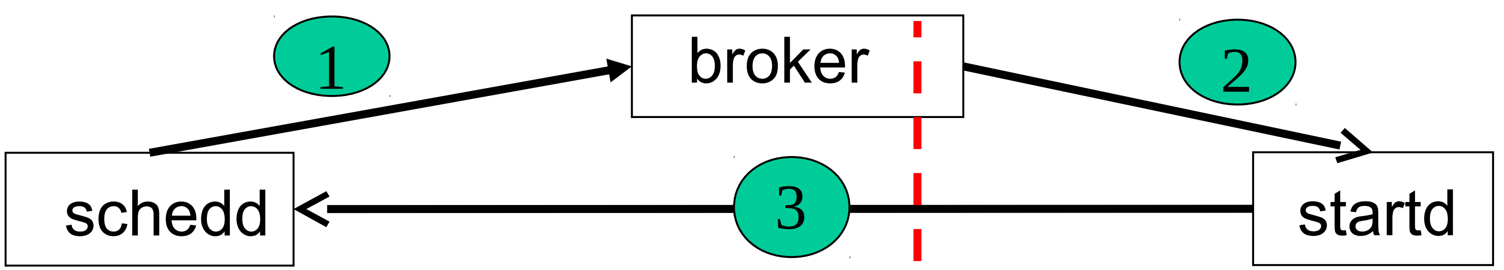
- Flocking
  - Metascheduling: connect scheduler to several condor pools
- Grid interface
- Pilot jobs
  - Job-based reservation of resources and application level scheduling

Flocking

- Submit a scheduler to several pools
  - Share condor pools between institutions
  - Try to run on local pool first, then try to run on remote pool

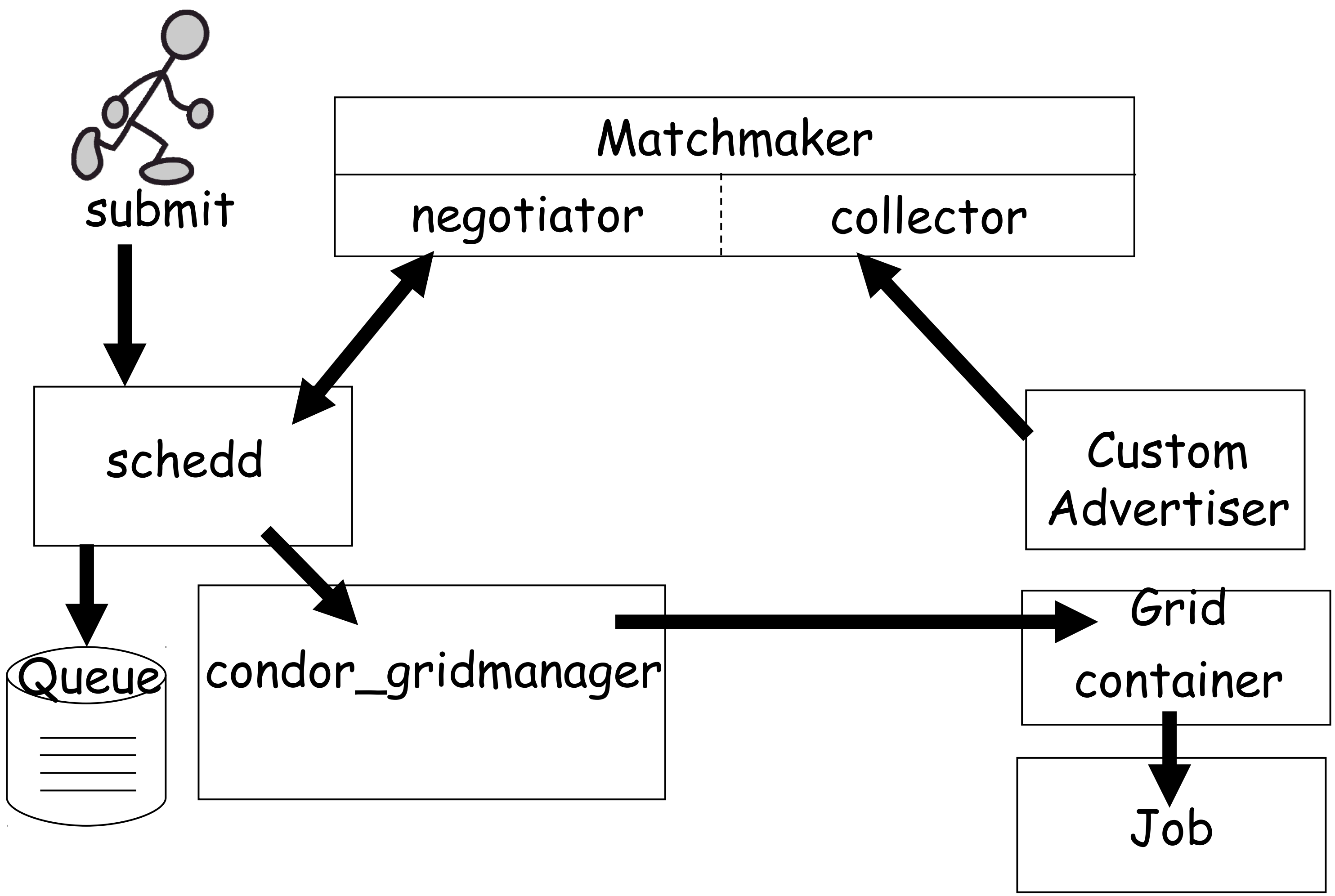


- Networking issues with private networks
  - A communication broker may be needed if the scheduler is not able to communicate directly with every execute node



Condor-G

- Submit jobs to other grid systems
  - Minimal changes to job description
- Grids
  - Globus 2
  - Globus 4
  - Amazon EC2
  - Nordugrid
  - Unicore
  - PBS
  - LSF
  - Condor (!)
  - ...

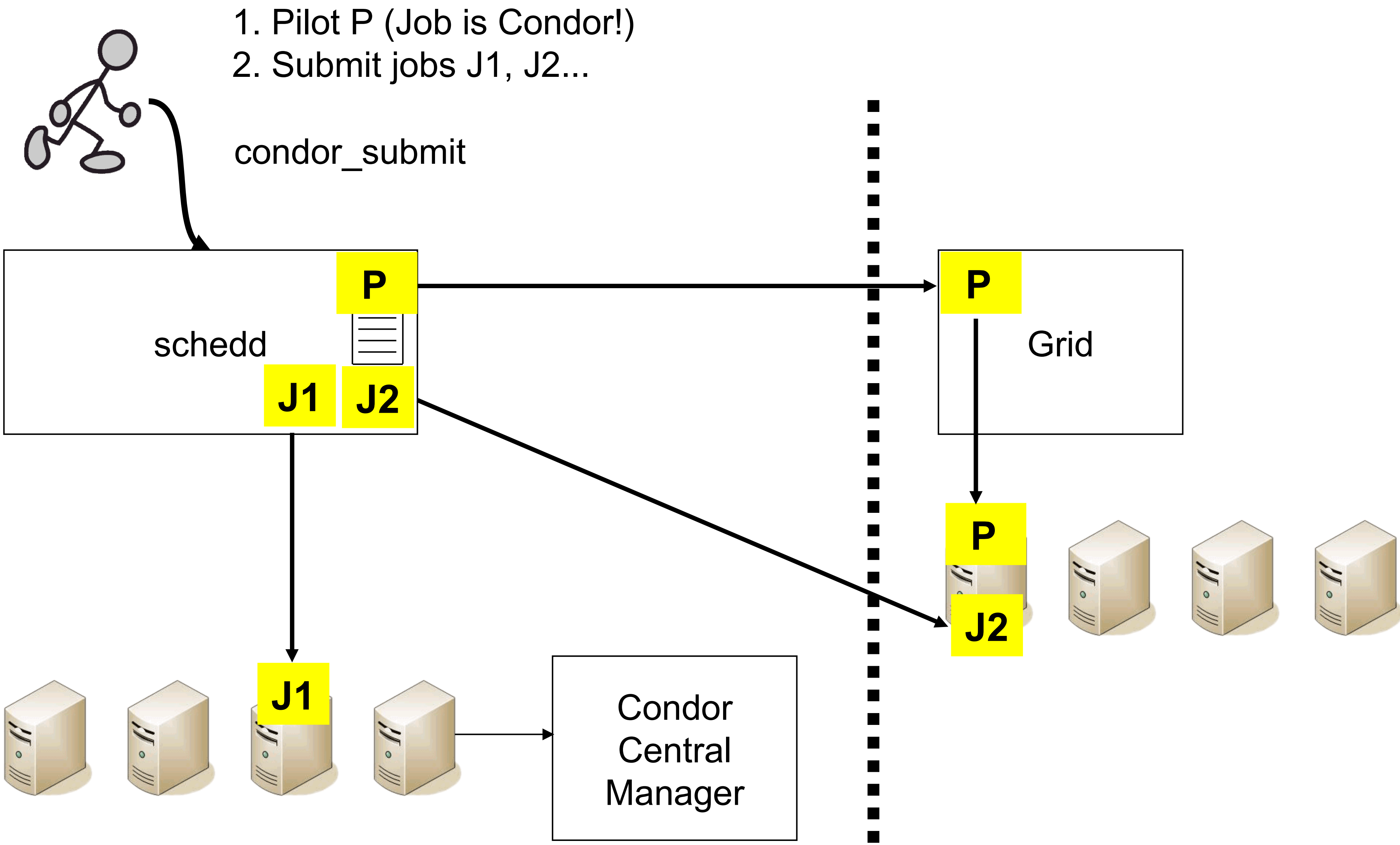


Pilot jobs

- Pilot jobs are application-level scheduler jobs that once executing on a grid resource schedule other jobs on it
  - Resources allocation through a custom batch system (bypasses grid workload manager)
  - Enables application-level scheduling
- Example: overlaying Condor on another system
  - Submit startd as a grid job to start a new pilot
  - Grow the condor pool with the new startd daemon
- Limitations
  - The scheduler has to be able to open communication with the pilot jobs
  - Security is tricky (whose job is ran by the pilot?)
  - System administrators do not like pilots so much

Condor pilot jobs

- Startd can be rub as a grid pilot job (Condor Glide-in)



From Pilots to Clouds

- Clouds: resources allocation
- Pilots: resource dedication through job submission