Orchestration d'expériences à l'aide de processus métier

Tomasz Buchert

Directeurs de thèse

Lucas Nussbaum

Jens Gustedt

Contexte : la recherche sur les systèmes distribués

Réaliser des expériences peut être vraiment frustrant.

Car les expériences avec des systèmes distribués sont :

- pénibles
- difficiles à faire correctement
- complexes et incompréhensibles
- sensibles aux pannes

Automatisation d'expériences

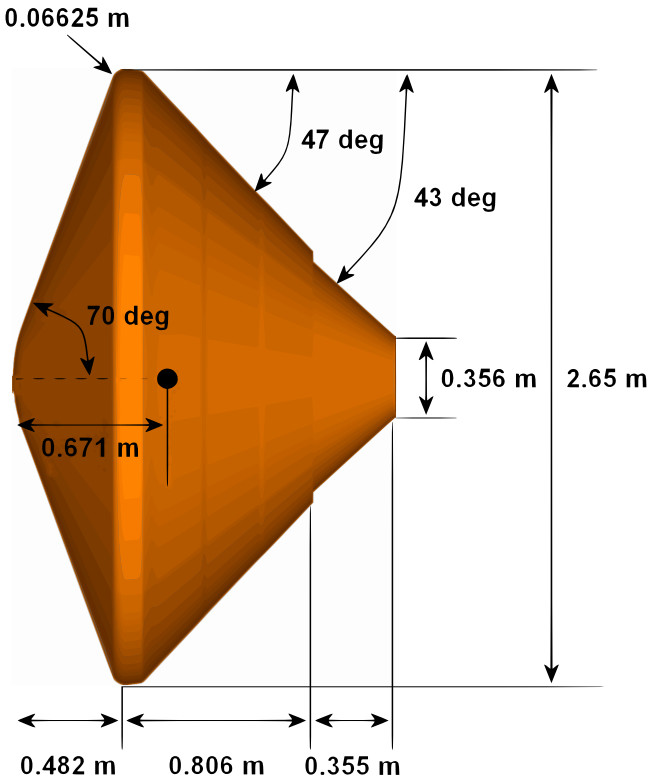
On doit **automatiser les expériences** :

- les humains font des erreurs
- les humains oublient
- le savoir se dissipe avec le temps

"*The primary decelerator for Phoenix was a rigid capsule with a 70-degree half-angle sphere-cone forebody (...)*"

Pourquoi cet angle exactement ?

Une simulation montrait qu'il est *optimal*.



Automatisation de l'administration système

Avec des outils comme Puppet et Chef :

- le facteur humain est éliminé
- les systèmes sont construits à partir de modules
- la configuration est reproductible

Pourtant, la reproductibilité n'implique pas la **descriptivité**.
Elle n'implique pas non plus la **facilité de compréhension**.

Outils pour l'expérimentation

Il existe un multitude d'outils pour la gestion d'expériences :

- Expo
- g5k-campaign
- OMF
- Plush
- ... et d'autres

Ils se basent sur des paradigmes différents.

Grid'5000 : Bash, oarsub, kadeploy3, ...

```
[ rennes ~ ] $ oarsub -I -l nodes=2 -t deploy
[ADMISSION RULE] Set default walltime to 3600.
[ADMISSION RULE] Modify resource description with type constraints
Generate a job key...
OAR_JOB_ID=454195
Interactive mode : waiting...
Starting...
Connect to OAR job 454195 via the node frontend.rennes.grid5000.fr
[ 454195@rennes ~ ] $ kadeploy3 -e squeeze-x64-nfs -f $OAR_FILE_NODES -k
Launching a deployment ...
Grab the key file /home/tbuchert/.ssh/authorized_keys
Performing a deploy step on the nodes: paradent-[17-18].rennes.grid5000.fr
--- switch_pxe (paradent cluster)
  >>>  paradent-[17-18].rennes.grid5000.fr
--- set_vlan (paradent cluster)
  >>>  paradent-[17-18].rennes.grid5000.fr
  *** Bypass the VLAN setting
--- reboot (paradent cluster)
  >>>  paradent-[17-18].rennes.grid5000.fr

... and so on.
```

Répétitif et requiert une grande attention.

g5k-campaign

```
class MyEngine < Grid5000::Campaign::Engine

    set :environment, "lenny-x64-base"
    set :resources, "nodes=4"
    set :walltime, 7200

    on :install! do |env, *args|
      nodes = env[:nodes]
      ssh(nodes.first, "root") do |ssh|
        ssh.exec!("apt-get install whatever")
        ssh.sftp.upload!("my_engine/some/file", "/destination/path")
      end
      env
    end

    on :execute! do |env, *args|
      env[:nodes].each { |node|
        ssh(node, "root") do |ssh|
          ssh.exec!("./my-program")
        end
      }
      env
    end
  end
```

Gère une réservation, mais reste plutôt de bas niveau.

Expo

```
reserv = ExpoEngine::new(@connection)
reserv.site = [ "bordeaux", "lille", "luxembourg", "nancy", "sophia" ]
reserv.resources = [ "nodes=50", "nodes=10", "nodes=4", "nodes=4", "nodes=30" ]
reserv.name = "Expo Scalability"
reserv.walltime = 600

reserv.run!

sizes = [ 10, 20, 40, 50, 80, $all.length ]

$all.each_slice_array(sizes) do |nodes|

  task_mon = Task::new("hostname", nodes, " Monitoring #{nodes.length} nodes")
  10.times {
    id, res = task_mon.execute
    puts " #{res.length} : #{res.duration}"
  }

end

reserv.stop!
```

Des abstractions utiles, une implémentation efficace, mais sans langage spécifique évolué.

Gush (Plush)

```
<plush>
  <project name="simple">
    <software name="SimpleSoftwareName" type="none">
      <package name="Package" type="web">
          <path>http://sysnet.cs.williams.edu/~jeannie/software.tar</path>
          <dest_path>software.tar</dest_path>
      </package>
    </software>
    <component name="Cluster1">
      <rspec><num_hosts>2</num_hosts></rspec>
      <resources><resource type="ssh" group="local"/></resources>
    </component>
    <experiment name="simple">
      <execution>
        <component_block name="cb1">
          <component name="Cluster1" />
          <process_block name="p2">
            <process name="cat">
              <path>cat</path><cmdline><arg>software.txt</arg></cmdline><cwd/>
            </process>
          </process_block>
        </component_block>
      </execution>
    </experiment>
  </project>
</plush>
```

Déclaratif, mais assez difficile à comprendre.

```
defGroup('Sender', "omf.nicta.node28") do |node|
  node.addApplication("test:app:otg2") do |app|
    app.setProperty('udp:local_host', '192.168.0.2')
    app.setProperty('udp:dst_host', '192.168.0.3')
    app.setProperty('udp:dst_port', 3000)
    app.measure('udp_out', :samples => 1)
  end
  w0 = node.net.w0; w0.mode = "adhoc"; w0.type = 'g'
  w0.channel = "6"; w0.essid = "helloworld"; w0.ip = "192.168.0.2"
end

defGroup('Receiver', "omf.nicta.node29") do |node|
  node.addApplication("test:app:otr2") do |app|
    app.setProperty('udp:local_host', '192.168.0.3')
    app.setProperty('udp:local_port', 3000)
    app.measure('udp_in', :samples => 1)
  end
  w0 = node.net.w0; w0.mode = "adhoc"; w0.type = 'g'
  w0.channel = "6"; w0.essid = "helloworld"; w0.ip = "192.168.0.3"
end

onEvent(:ALL_UP_AND_INSTALLED) do |node|
  info "This is my first OMF experiment"; wait 10; allGroups.startApplications
  info "All my Applications are started now..."; wait 30; allGroups.stopApplications
  info "All my Applications are stopped now."; Experiment.done
end
```

Déclaratif et fondé sur les événements.

« Bottom-up » vs « top-down »

Les outils présentés utilisent un **design** « **bottom-up** ».

Peut-on renverser le problème avec une **approche** « **top-down** » ?

1. Commencer avec une description de haut niveau.
2. Implanter les détails de bas niveau.
3. Lancer l'expérience.
4. Améliorer (si nécessaire) et itérer.

Il existe une méthode exactement comme ça.

Business Process Management

Le *Business Process Management* vise à :

- comprendre une organisation
- modélisation ses processus comme des **workflows**
- **gérer** ces processus et les **surveiller**
- **améliorer** les **activités** de l'organisation
- réimplanter les processus pour les rendre :
  - moins chers
  - moins propices aux pannes
  - plus rapides et efficaces

XPflow

Dans cette présentation je vais présenter **XPflow** :

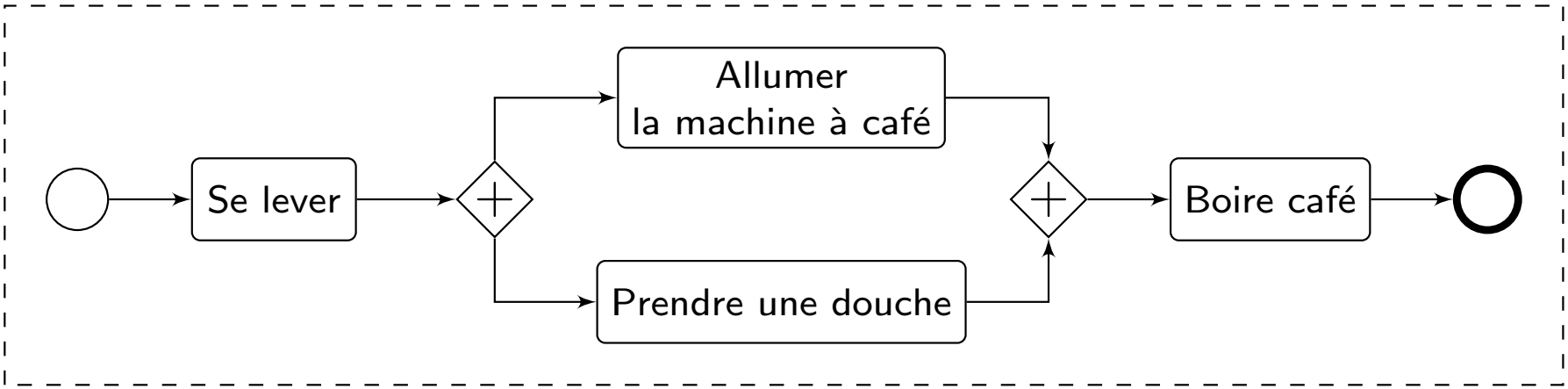
- un nouveau moteur de conduite d'expériences
- basé sur le **Business Process Modeling** et **Management**

Il sera illustré à l'aide d'expériences effectuées sur Grid'5000.

Concepts d'XPflow

XPflow introduit 2 concepts principaux :

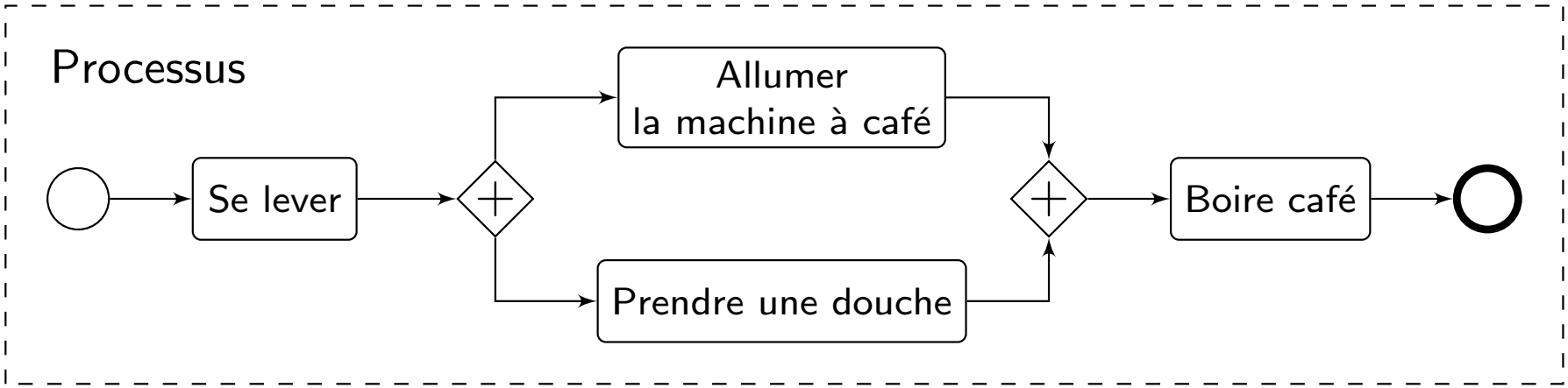
- Processus – une description de haut niveau :
  - décrivent les workflows dans langage dédié (DSL)
  - gèrent les autres processus et activités
- Activités – l'expérience de bas niveau :
  - font le vrai travail
  - écrits dans un langage de programmation classique



Concepts d'XPflow

XPflow introduit 2 concepts principaux :

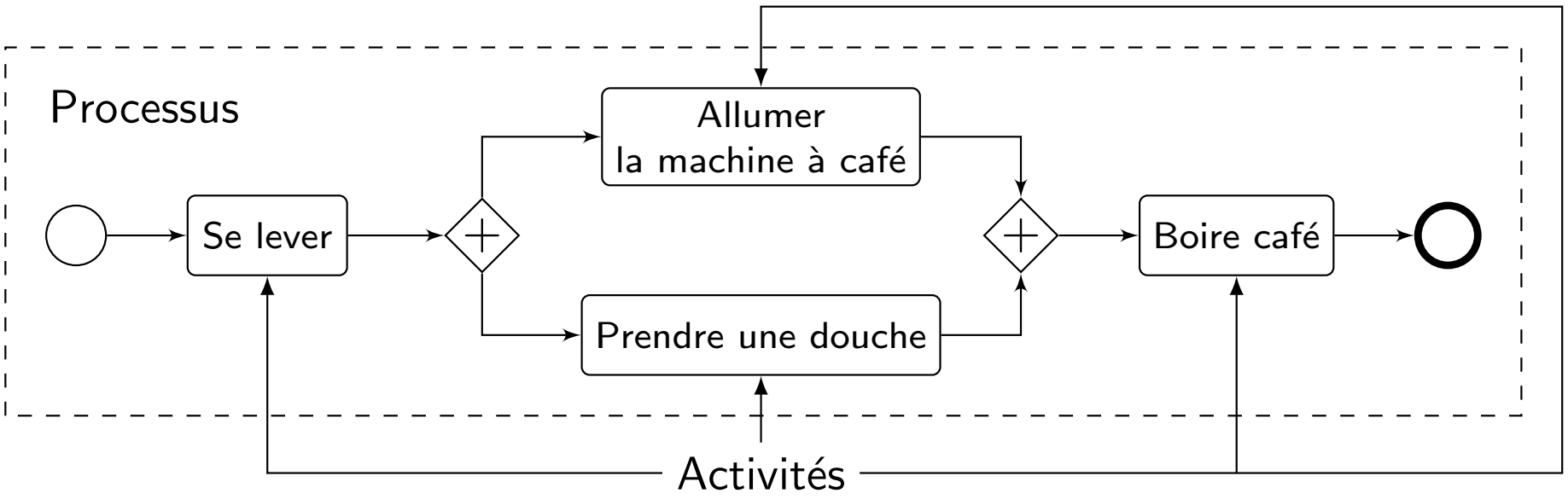
- Processus – une description de haut niveau :
  - décrivent les workflows dans langage dédié (DSL)
  - gèrent les autres processus et activités
- Activités – l'expérience de bas niveau :
  - font le vrai travail
  - écrits dans un langage de programmation classique



Concepts d'XPflow

XPflow introduit 2 concepts principaux :

- Processus – une description de haut niveau :
  - décrivent les workflows dans langage dédié (DSL)
  - gèrent les autres processus et activités
- Activités – l'expérience de bas niveau :
  - font le vrai travail
  - écrits dans un langage de programmation classique



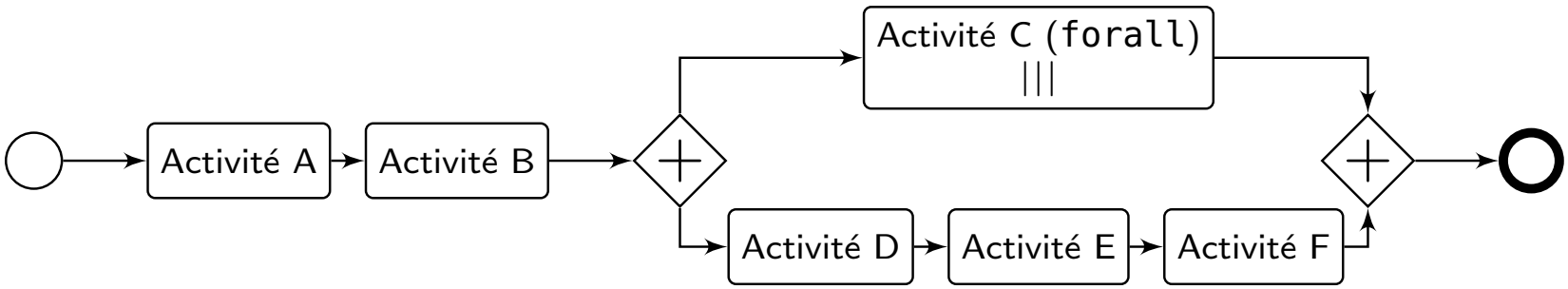
Langage dédié

Le langage dédié contient **des primitives spécifiques** pour :

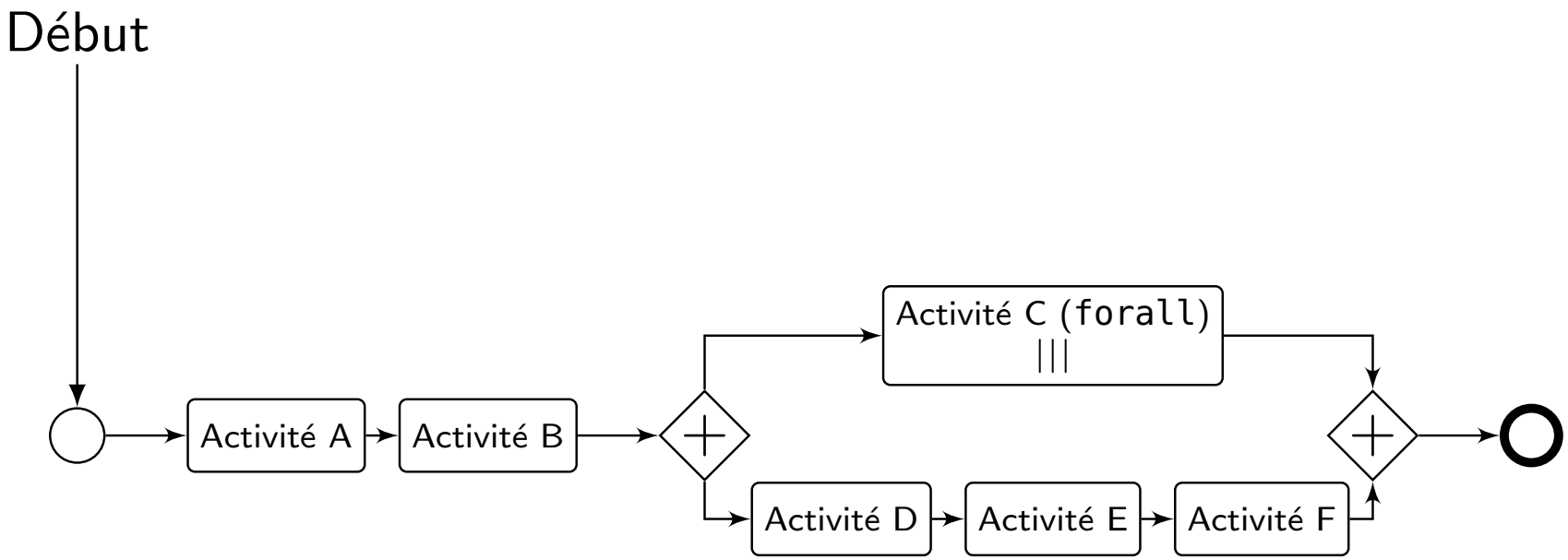
- lancer des activités et des sous-processus (`run`),
- lancer des activités séquentiellement ou en parallèle (`sequence`, `parallel`),
- exécuter des instructions conditionnelles (`if`, `switch`)
- lancer des boucles séquentielles ou parallèles (`loop`, `foreach`, `forall`),
- gérer les erreurs (`try`, `checkpoint`).

Certaines de ces primitives sont empruntées directement au BPM.

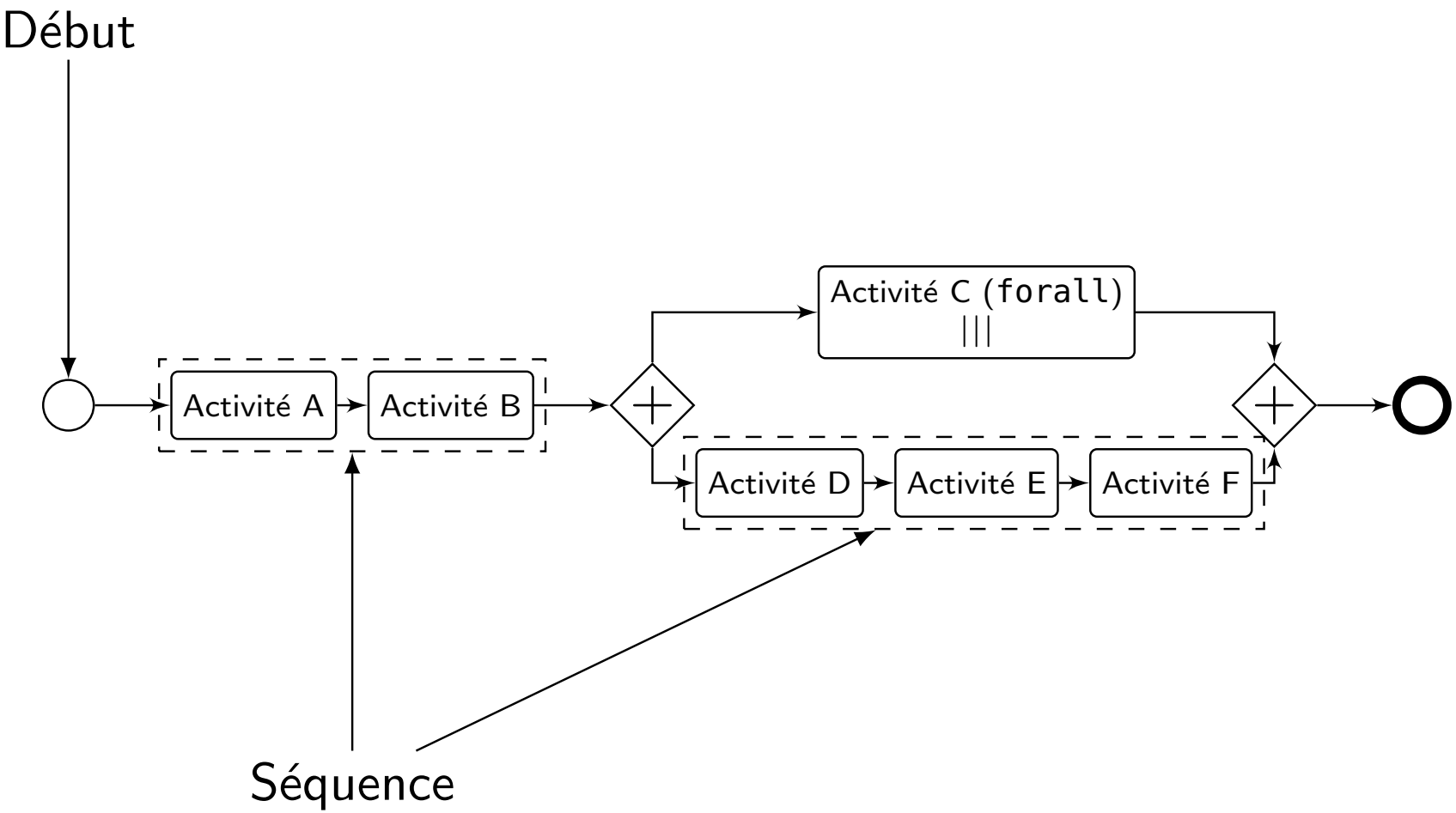
Primitives workflow (exemple)



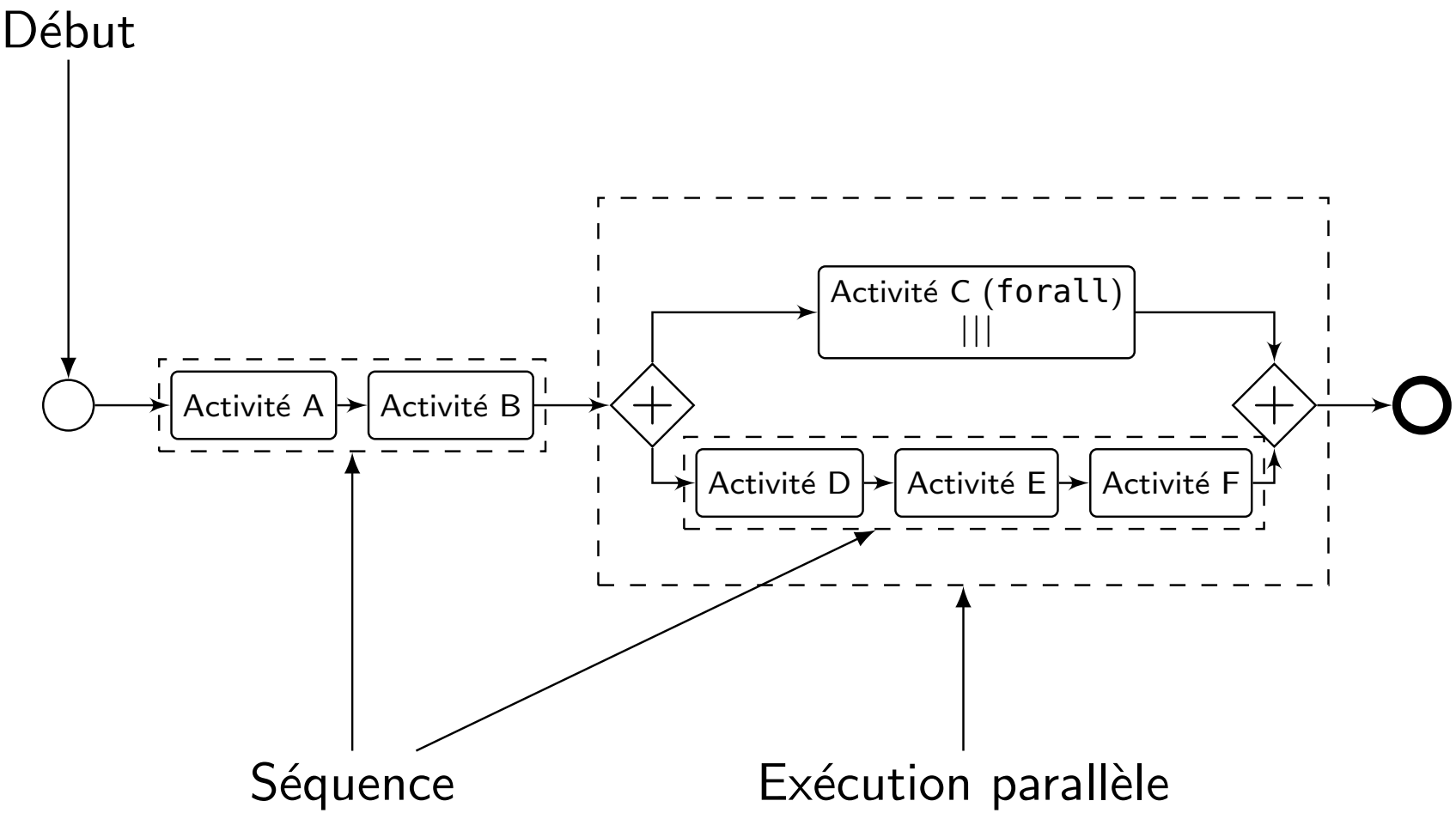
Primitives workflow (exemple)



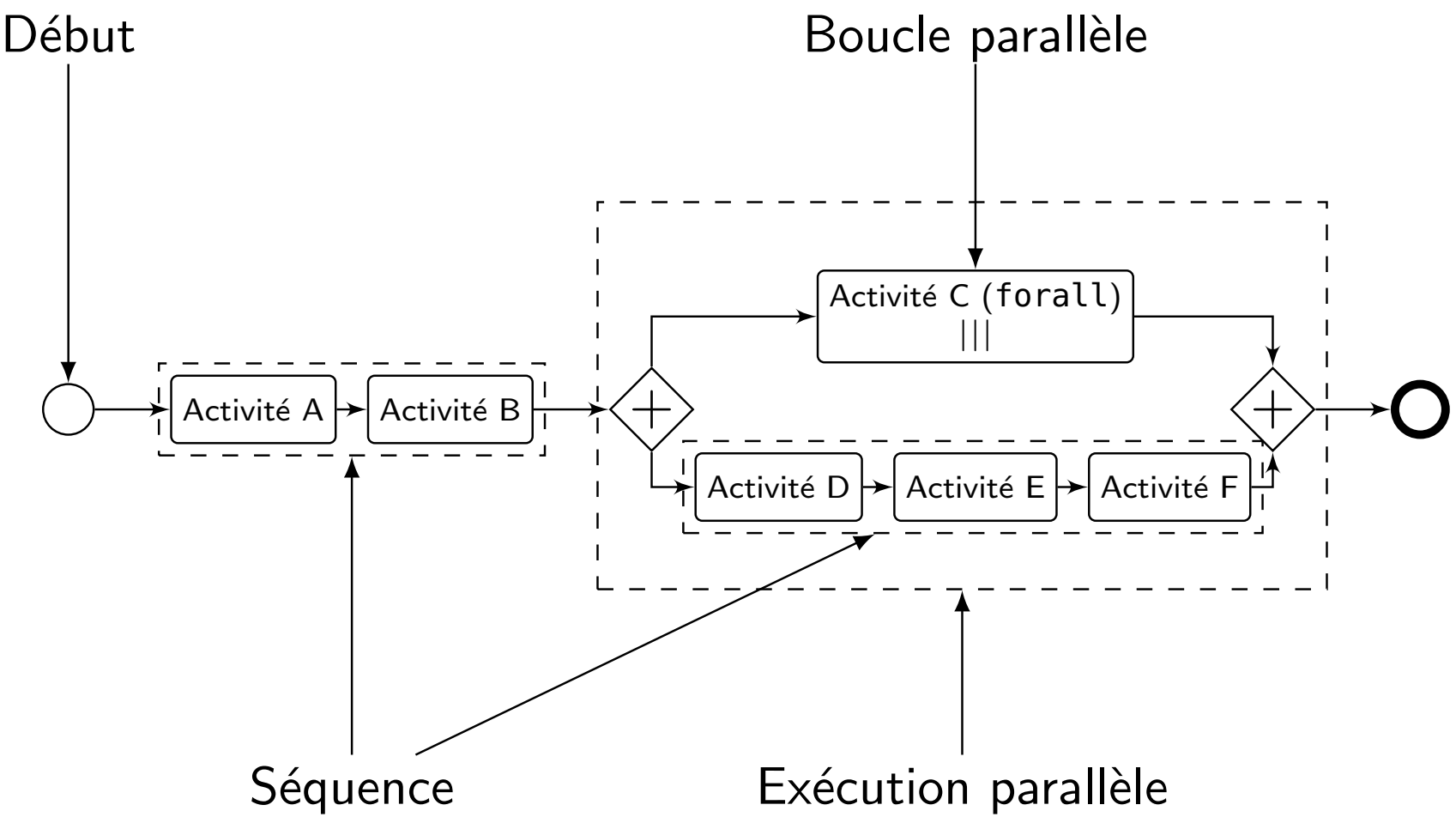
Primitives workflow (exemple)



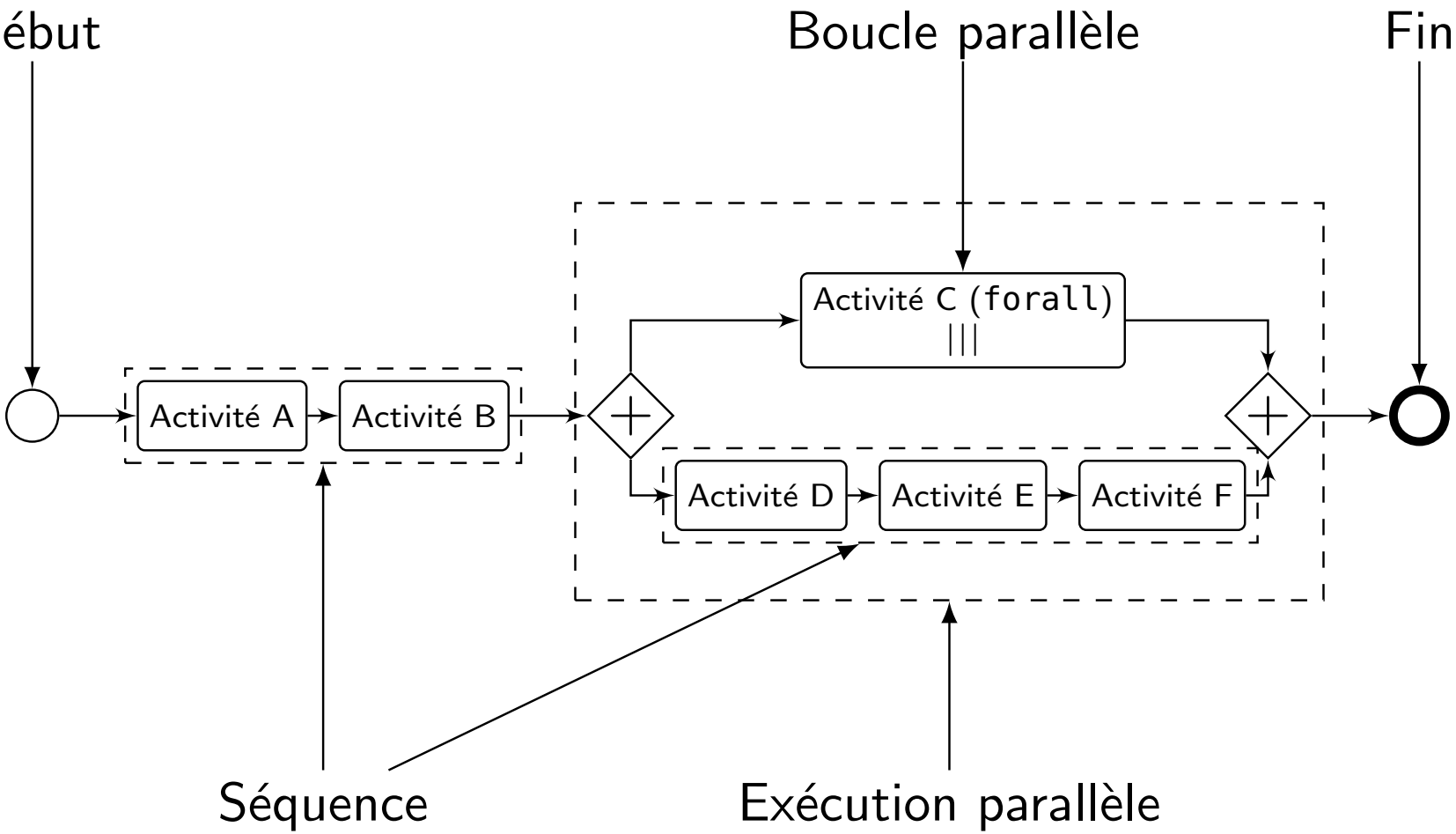
Primitives workflow (exemple)



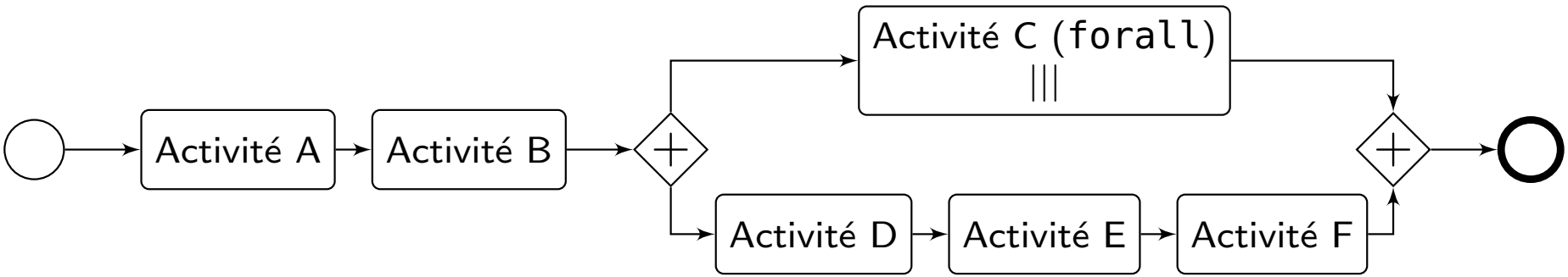
Primitives workflow (exemple)



Primitives workflow (exemple)

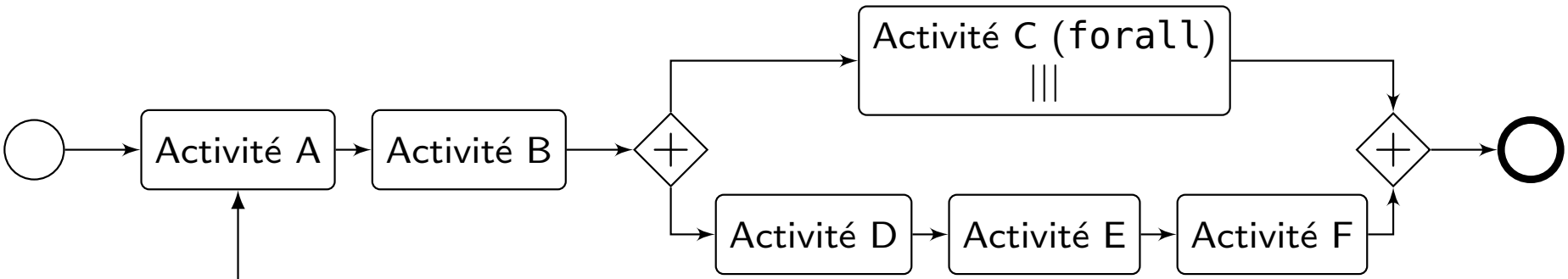


Primitives workflow (exemple, suite)



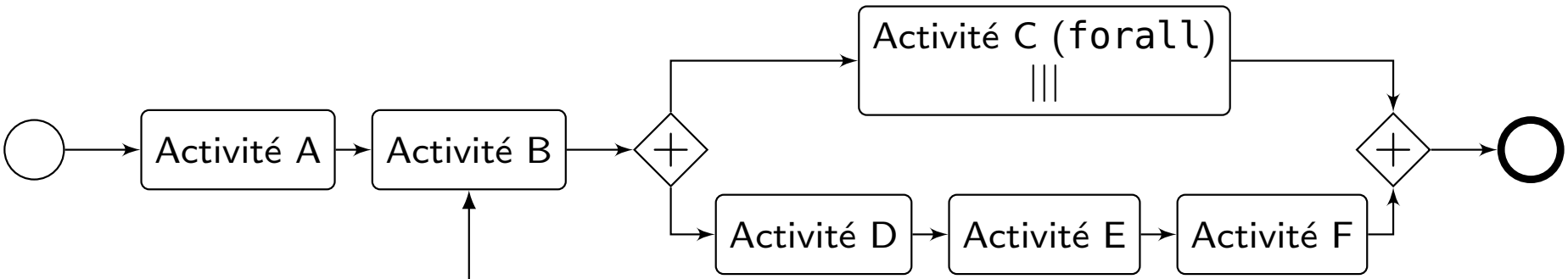

```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Primitives workflow (exemple, suite)



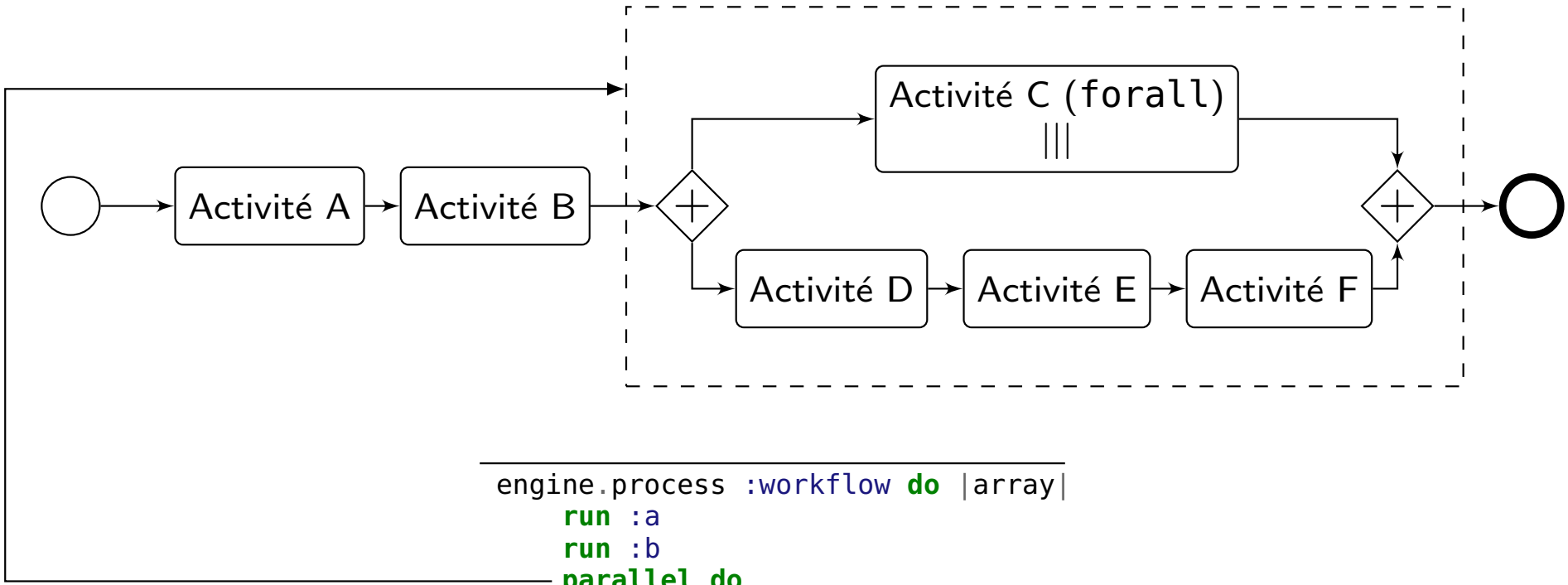

```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Primitives workflow (exemple, suite)



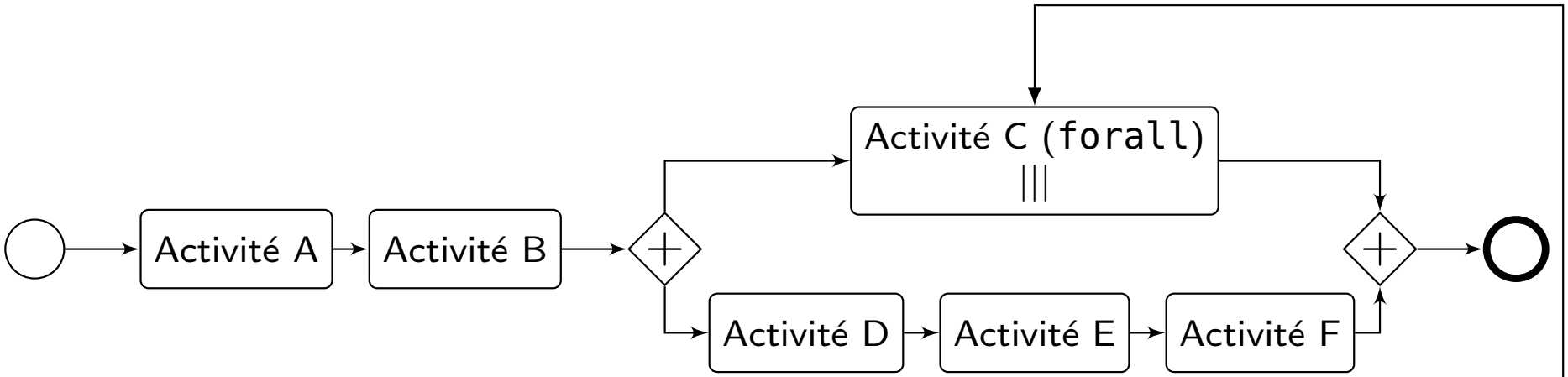

```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Primitives workflow (exemple, suite)



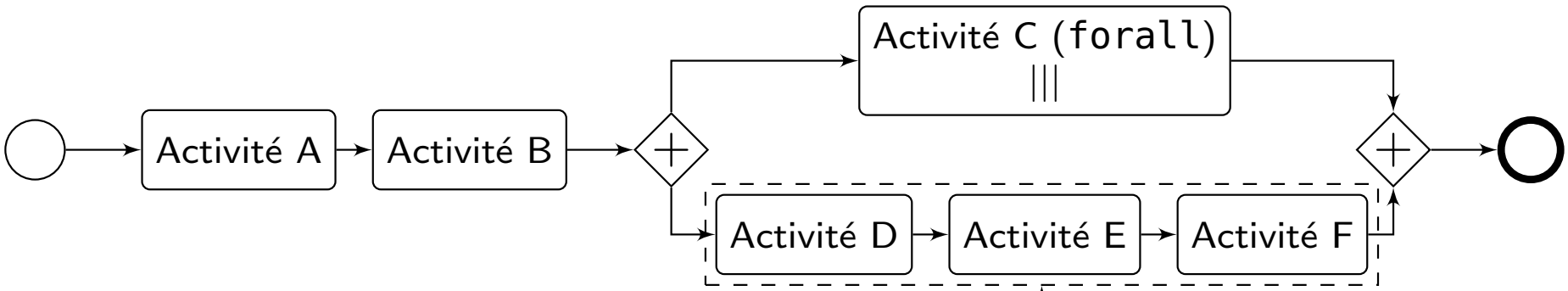

```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Primitives workflow (exemple, suite)



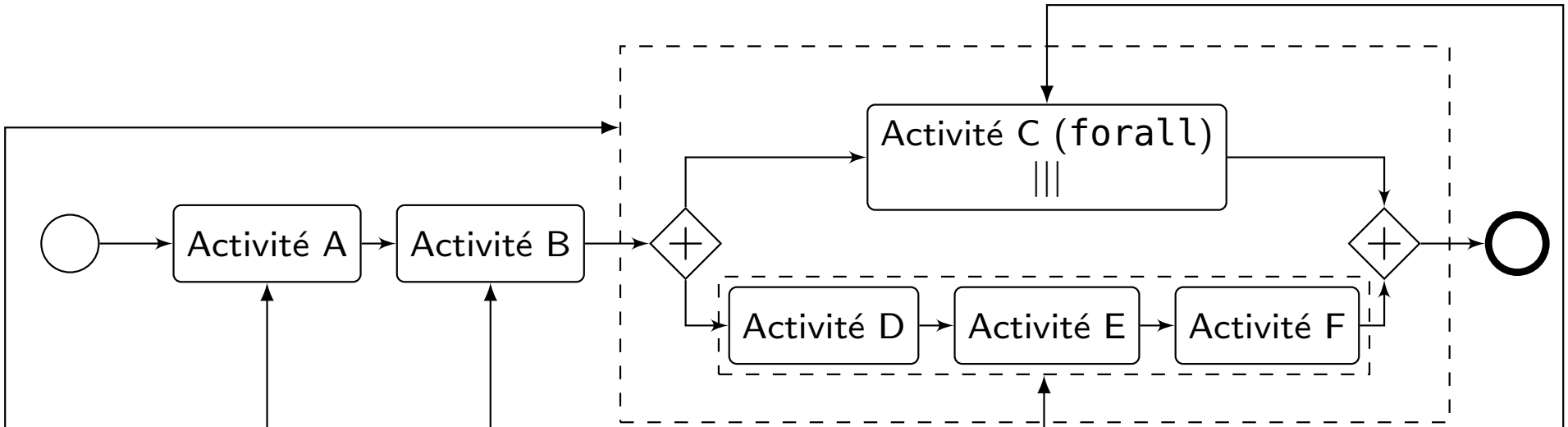

```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Primitives workflow (exemple, suite)




```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Primitives workflow (exemple, suite)




```
engine.process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Gestion des erreurs

XPflow propose les deux moyens principaux de gérer des erreurs :

- sauvegarde d'instantanés (*snapshotting*) :
  - sauvegarde l'état d'exécution
  - raccourcit le cycle de développement
- réessai (*retry policy*) :
  - en cas d'erreur, relance un sous-workflow
  - améliore la fiabilité

```
engine.process :snapshotting do
    run :long_deployment
    checkpoint :d
    run :experiment
end
```

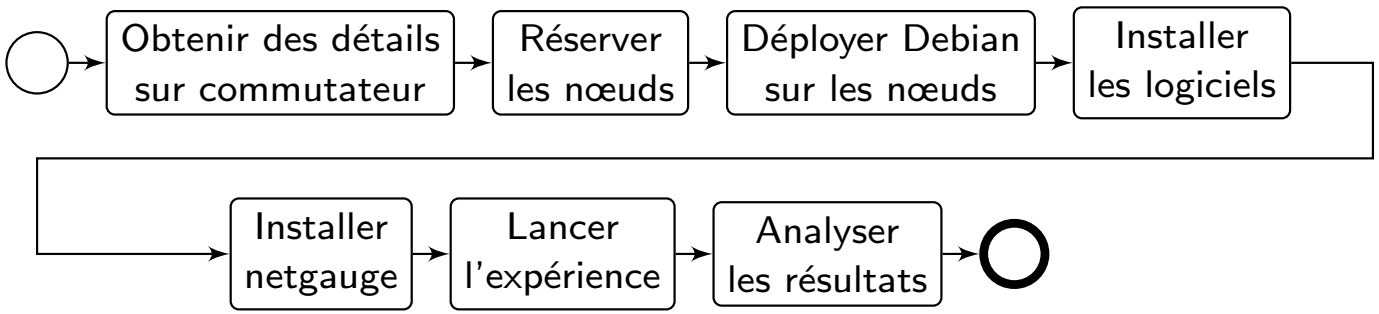
```
engine.process :retrying do
    try :retry => 5 do
        run :tricky_activity
    end
end
```

Exemple d'expérience

Mesure de l'*effective bisection bandwidth* d'un commutateur réseau.

1. Obtenir les noms des nœuds connectés au commutateur.
2. Réserver les nœuds.
3. Déployer Debian.
4. Installer les logiciels nécessaires.
5. Compiler et installer *netgauge*.
6. Lancer l'expérience elle-même.
7. Analyser les résultats.

Workflow d'expérience

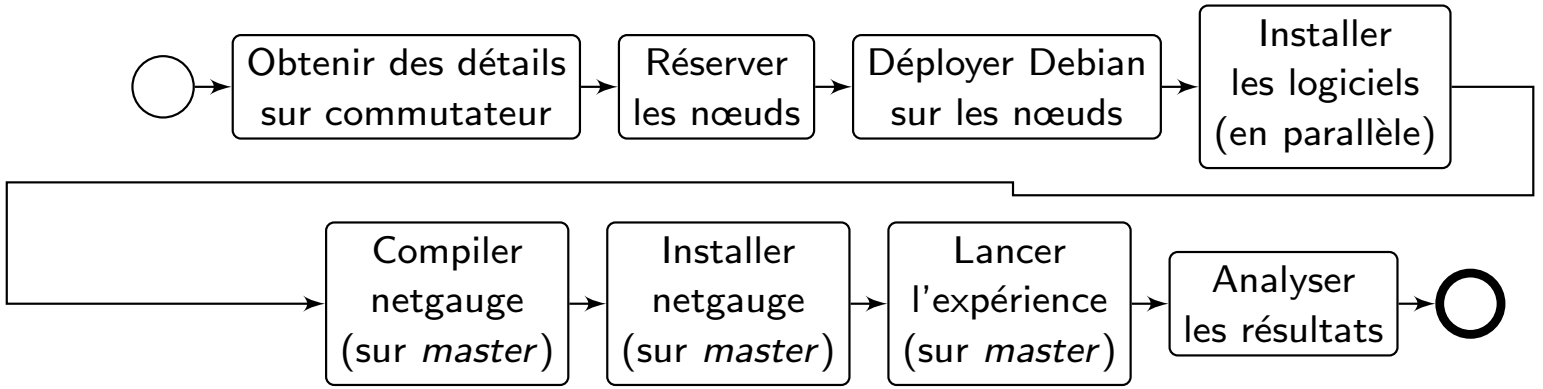


Quelques notes :

- chaque nœud doit avoir les logiciels installé
- chaque nœud doit avoir *netgauge* installé ...
- ... mais il suffit un nœud pour le compiler
- seulement un nœud lance l'application MPI

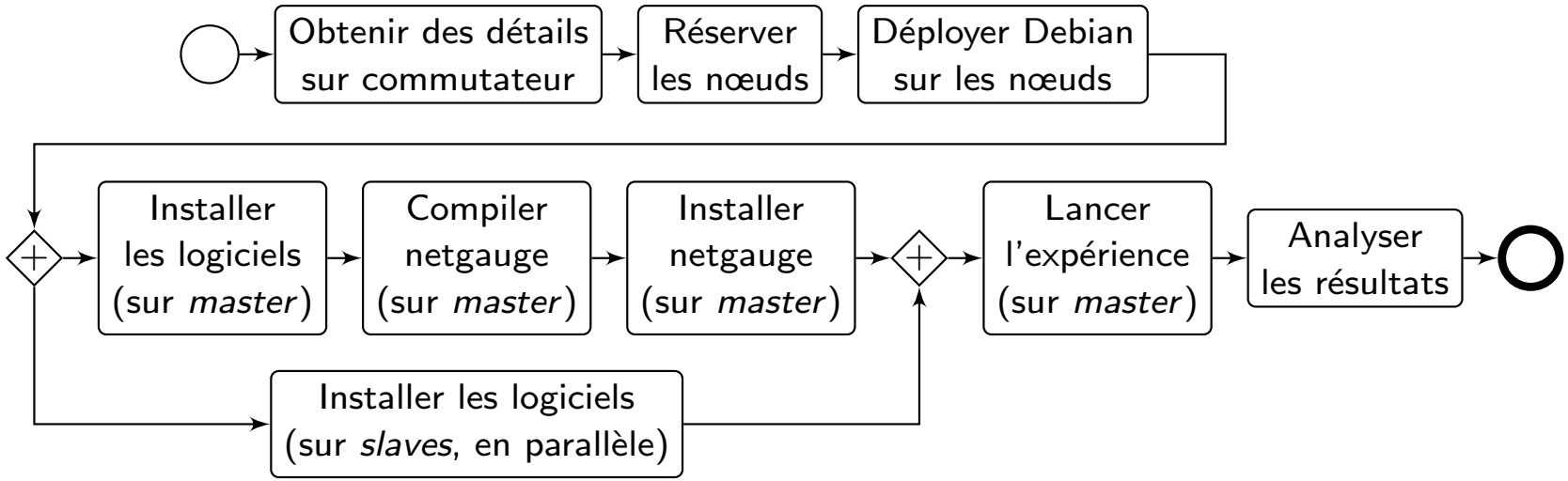
Il faut répartir les nœuds en *master* et *slaves*.

Workflow d'expérience



Encore une observation : la compilation peut être en parallèle avec l'installation du logiciel sur les *slaves*.

Workflow d'expérience



C'est un workflow qui décrit l'expérience.
Puis, on va l'exprimer dans XPflow.

Workflow d'expérience en DSL

```
engine.process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Workflow d'expérience en DSL

```
engine.process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activité :install_pkgs

```
engine.activity :install_pkgs do|node|
    log 'Installing packages on ', node
    run 'g5k.bash', node do
        aptget :update
        aptget :upgrade
        aptget :purge, 'mx'
    end
end
```

Workflow d'expérience en DSL

```
engine.process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activité :build_netgauge

```
engine.activity :build_netgauge do |master|
    log "Building netgauge on #{master}"
    run 'g5k.copy', NETGAUGE, master, '~'
    run 'g5k.bash', master do
        build_tarball NETGAUGE, PATH
    end
    log "Build finished."
end
```

Workflow d'expérience en DSL

```
engine.process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activité :dist_netgauge

```
engine.activity :dist_netgauge do |m, s|
    master, slaves = m, s
    run 'g5k.dist_keys', master, slaves
    run 'g5k.bash', master do
        distribute BINARY,
            DEST, 'localhost', slaves
    end
end
```

Workflow d'expérience en DSL

```
engine.process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activité :netgauge

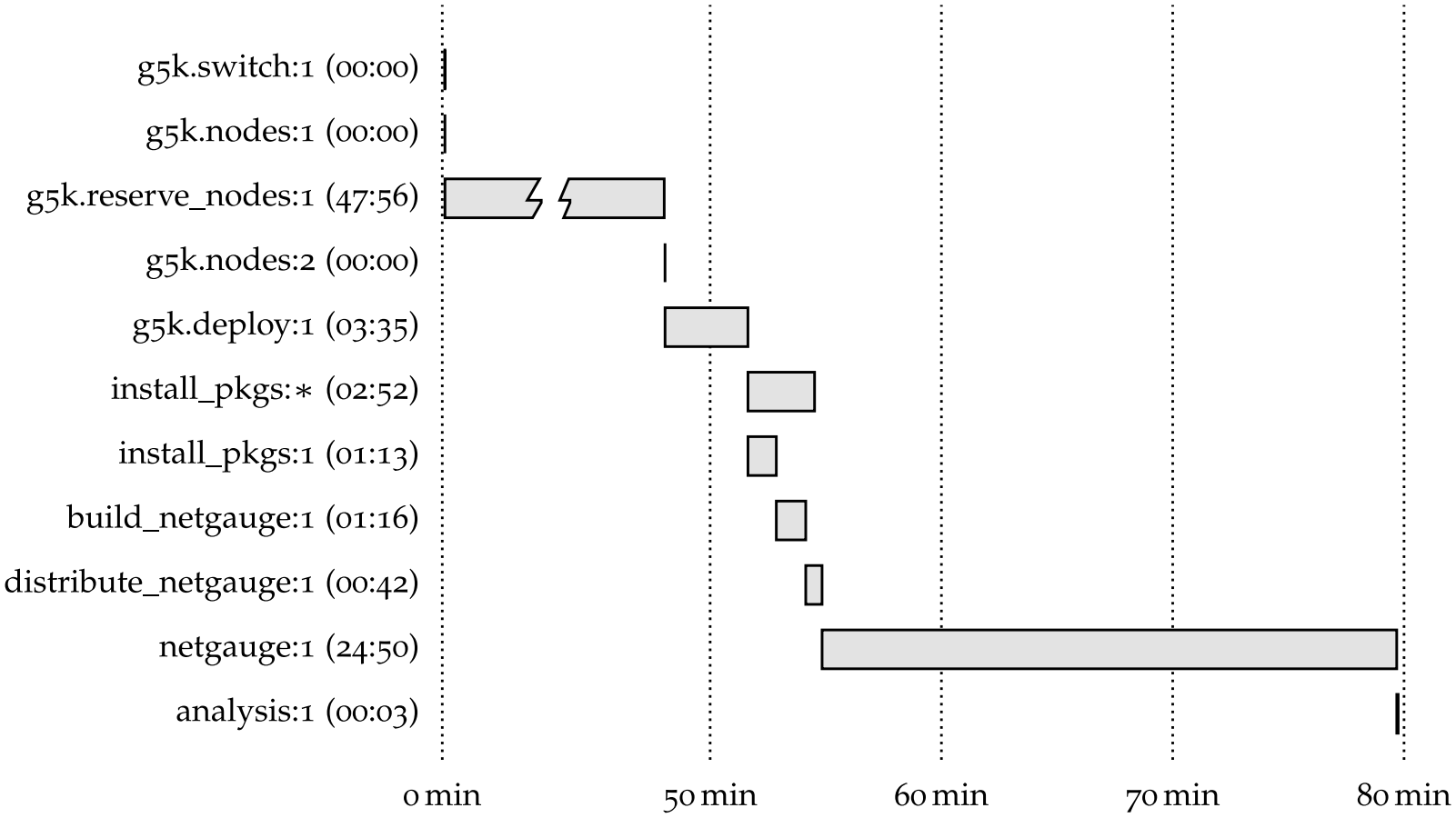
```
engine.activity :netgauge do |master, nodes|
    log "Running experiment..."
    out = run 'g5k.bash', master do
        cd PATH
        mpirun nodes, "./netgauge"
    end
    log "Experiment done."
end
```

Exemple d'exécution

```
[ 11:15:52.940 ] Started activity g5k.switch:1.
[ 11:15:53.418 ] Finished activity g5k.switch:1 (0.478 s).
[ 11:15:53.419 ] Process exp: Experimenting with switch: sgraphene2
[ 11:15:53.419 ] Started activity g5k.nodes:1.
[ 11:15:53.419 ] Finished activity g5k.nodes:1 (0.000 s).
[ 11:15:53.419 ] Started activity g5k.reserve_nodes:1.
[ 11:15:55.837 ] Waiting for reservation 408387
[ 11:16:02.452 ] Reservation 408387 should be available in 12 mins
[ 11:16:02.452 ] Reservation 408387 ready
[ 11:16:02.453 ] Finished activity g5k.reserve_nodes:1 (9.022 s).
[ 11:16:02.453 ] Started activity g5k.nodes:2.
[ 11:16:02.453 ] Finished activity g5k.nodes:2 (0.000 s).
[ 11:16:02.453 ] Started activity g5k.deploy:1.
[ 11:22:09.427 ] Finished activity g5k.deploy:1 (366.968 s).
[ 11:22:09.429 ] Started activity install_pkgs.
[ 11:22:09.429 ] Started activity install_pkgs:1.
[ 11:22:09.430 ] Activity install_pkgs: Installing packages on graphene-96
[ 11:22:09.430 ] Started activity install_pkgs:2.
[ 11:22:09.430 ] Activity install_pkgs: Installing packages on graphene-60
```

Le déroulement de l'exécution peut être minutieusement surveillé.

Monitoring - diagramme de Gantt

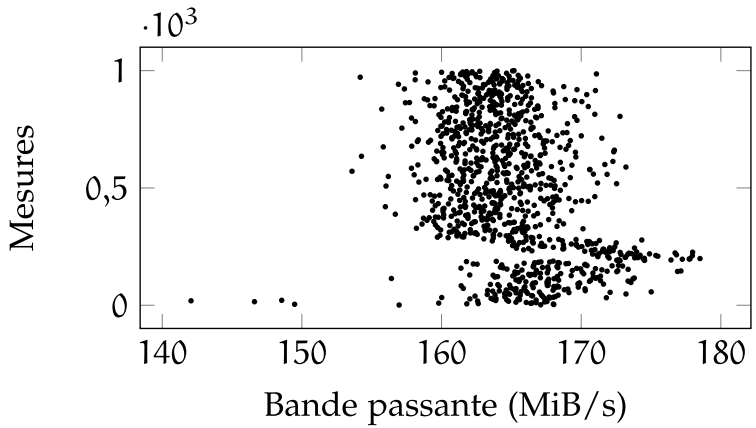


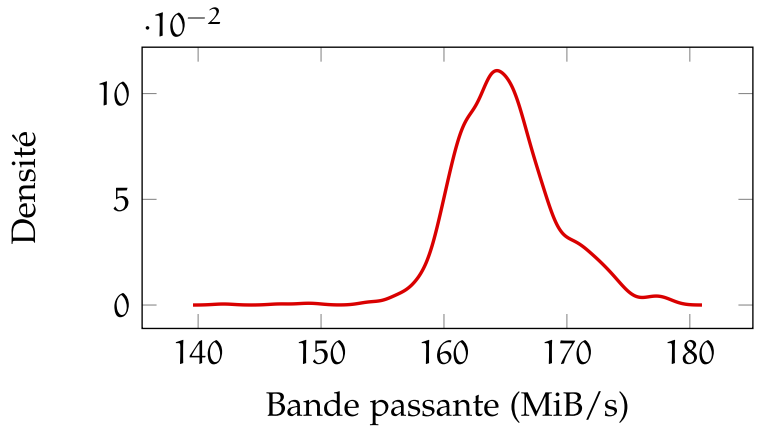
Chaque activité est surveillée pendant l'exécution.

Notamment, `build_netgauge:1` s'exécute en parallèle avec `install_pkgs:*`.

Résultat expérimental

Toute l'expérience est automatisée.
Cela inclut la génération de figures pour l'article.





Conclusions

Dans cette présentation, j'ai présenté XPflow.

XPflow introduit un paradigme novateur qui offre :

- l'expressivité,
- la modularité et la flexibilité,
- l'utilisation des concepts du BPM
- la surveillance du déroulement de l'expérience,
- l'intégration avec Grid'5000

Dans le futur, nous envisageons de travailler sur :

- l'intégration avec d'autres outils d'expérimentation
- l'interaction avec l'utilisateur pendant l'expérience
- la distribution efficace de données

Merci de votre attention. Questions ?