Plan

- **Storm**
- **Java-RMI**
- **Les Machines Réactives Distribuées**

Junior

Sémantique formelle
+
implémentations efficaces

implémentation de référence :

-Rewrite (Sémantique formelle SOS)

implémentation plus efficace :

-Replace (règle SOS dérivées de Rewrite)

grand nombre de composants parallèles et d'événements :

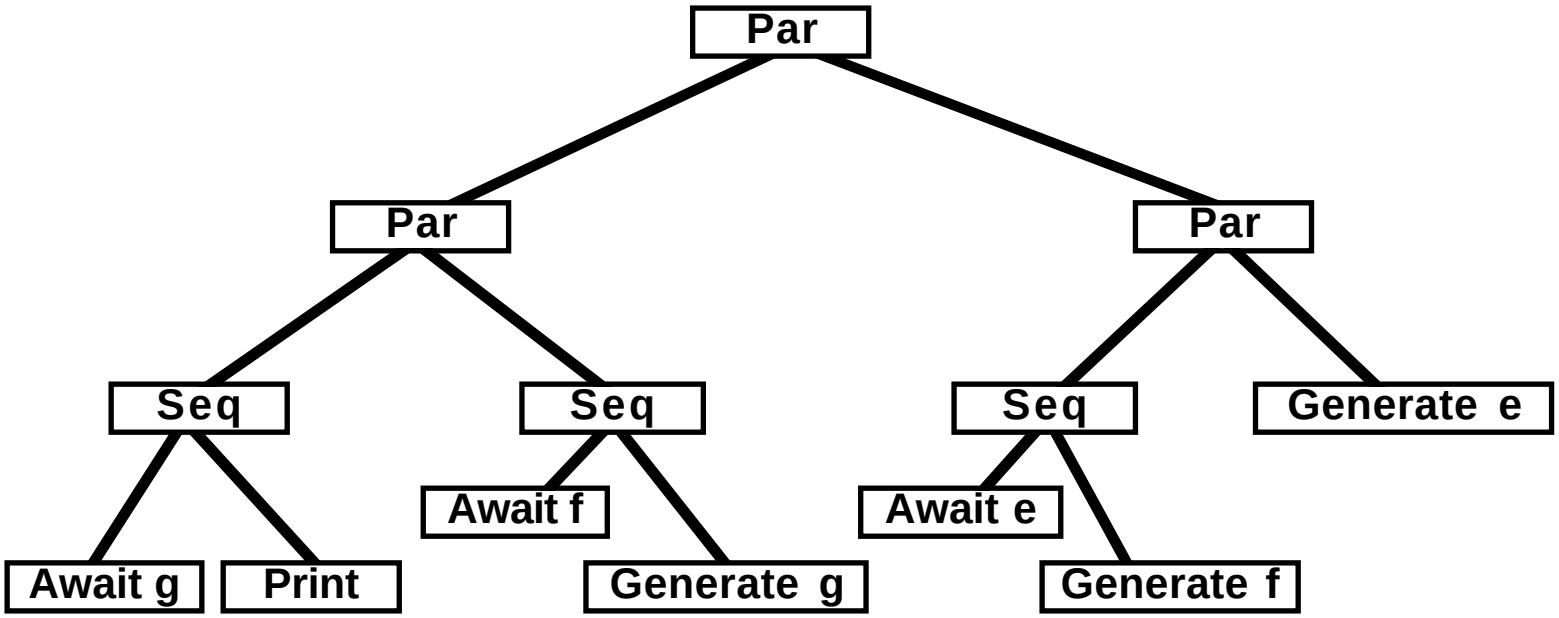
-Simple* (pas de sémantique formelle)

Rewrite -> Replace -> Storm -> Simple

***Simple est une implémentation de FT R&D écrite par L. Hazard**

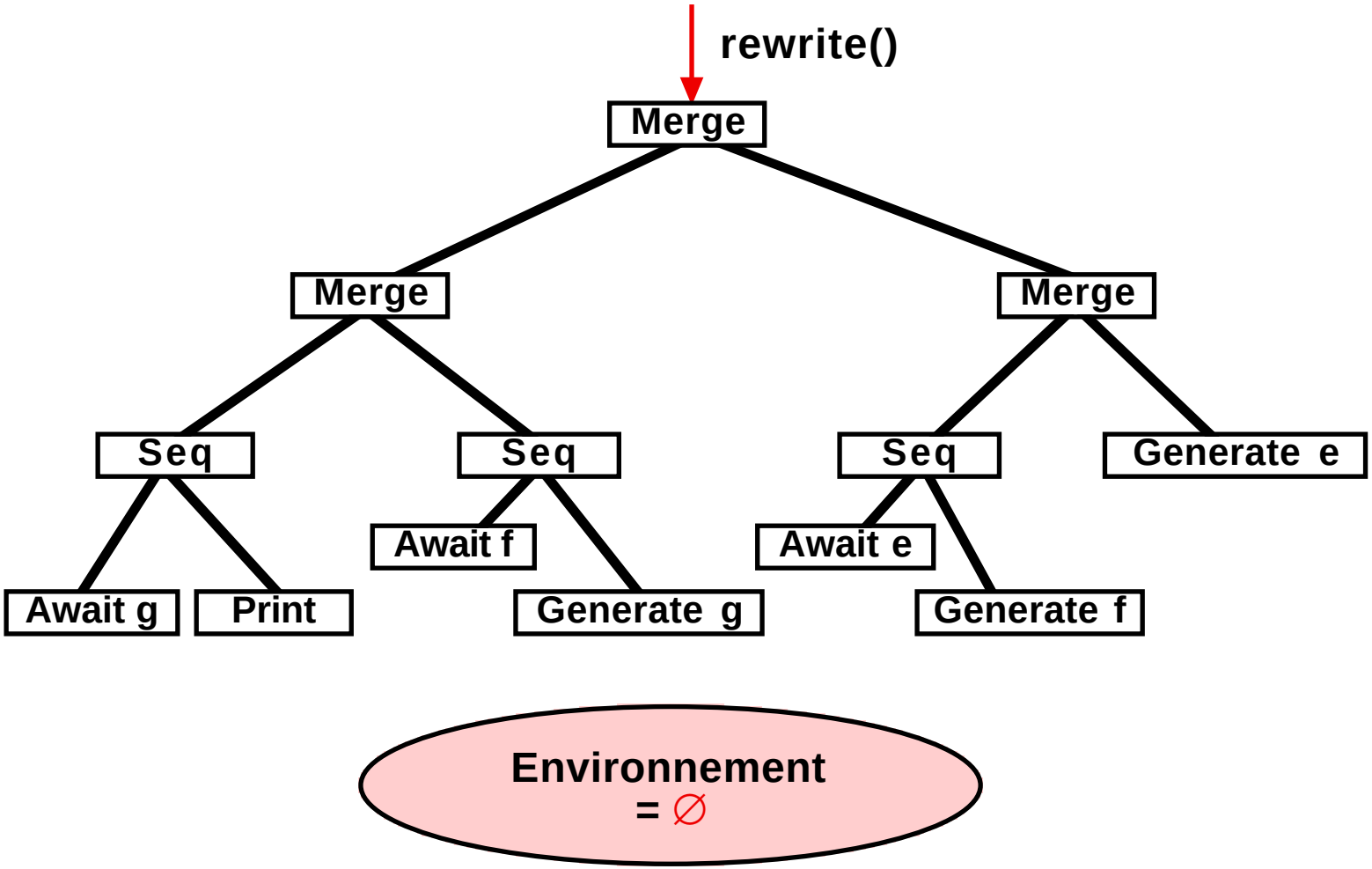
Implémentation efficace

Programme = Arborescence d'objet-instructions

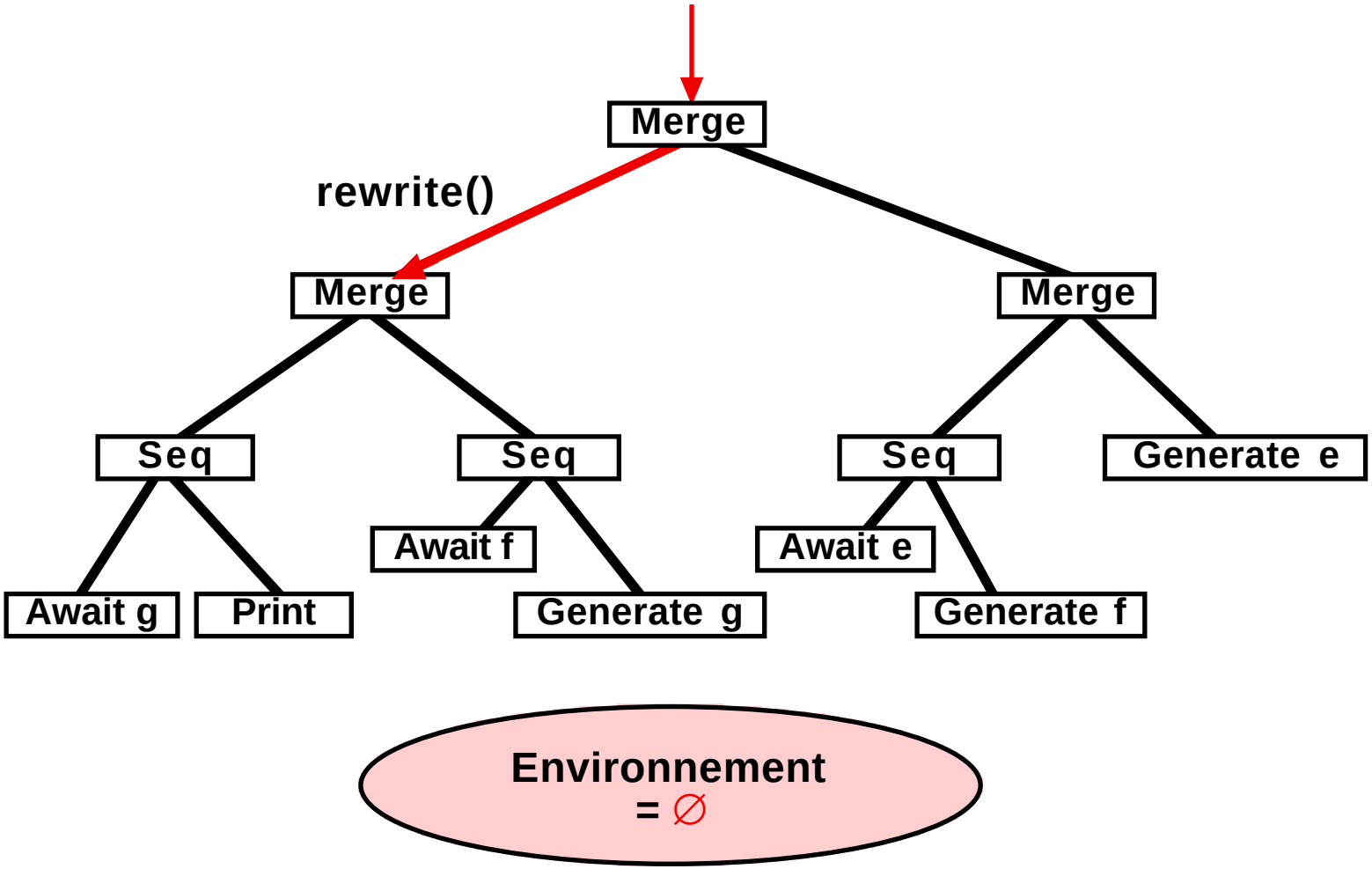


Les composants parallèles sont ordonnancés par la machine réactive selon la sémantique du parallélisme

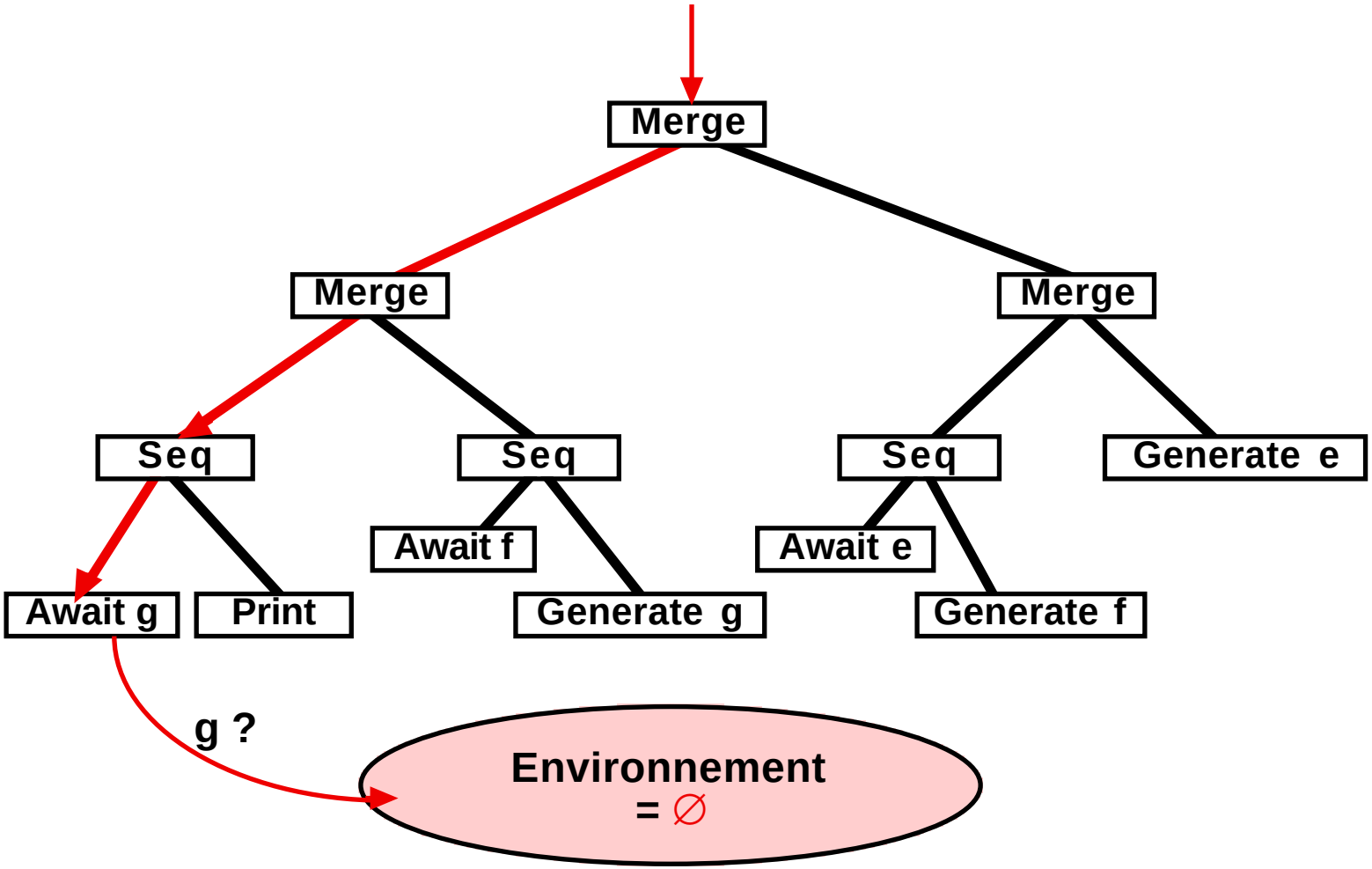
Rewrite-Replace



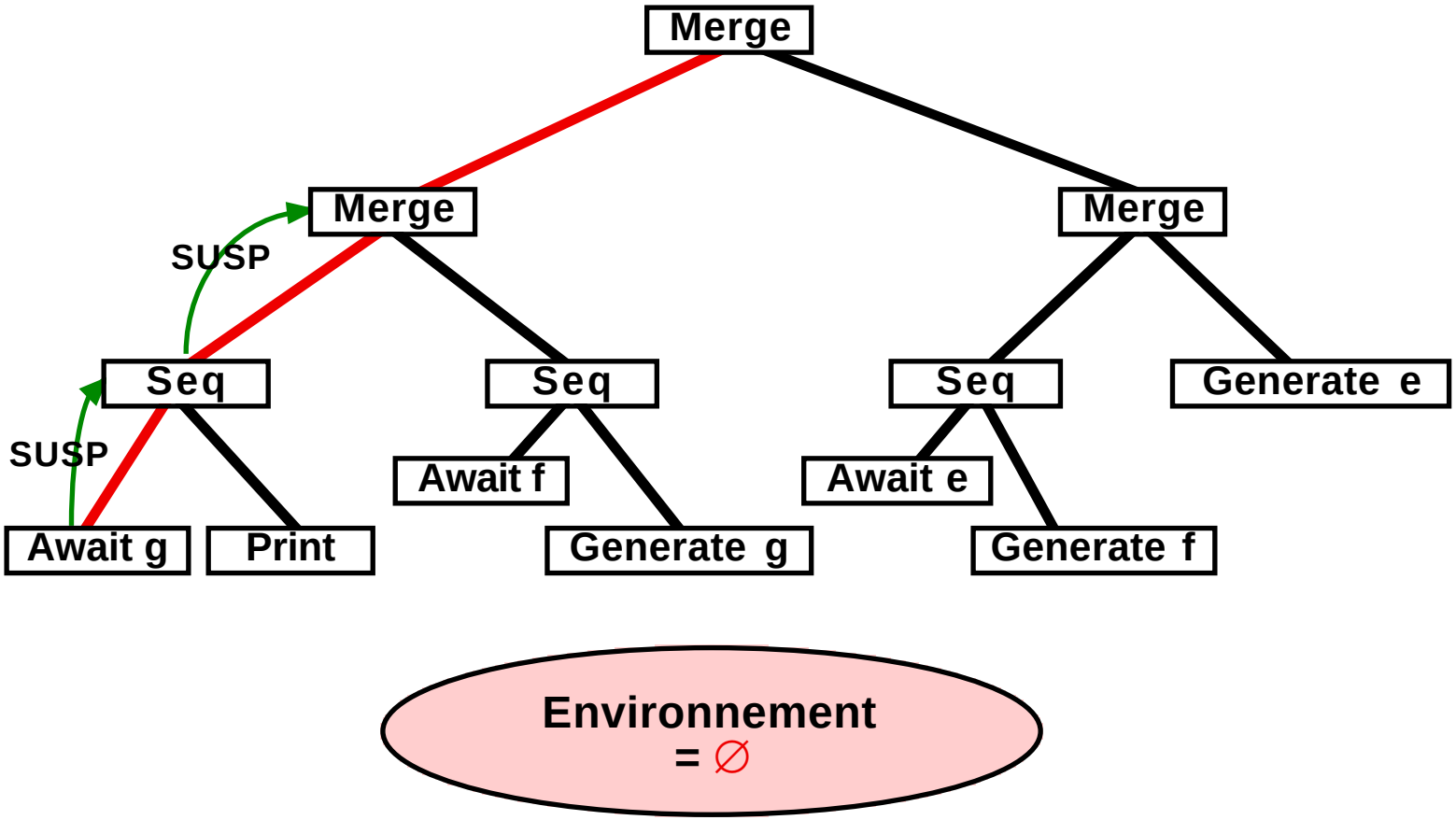
Rewrite-Replace



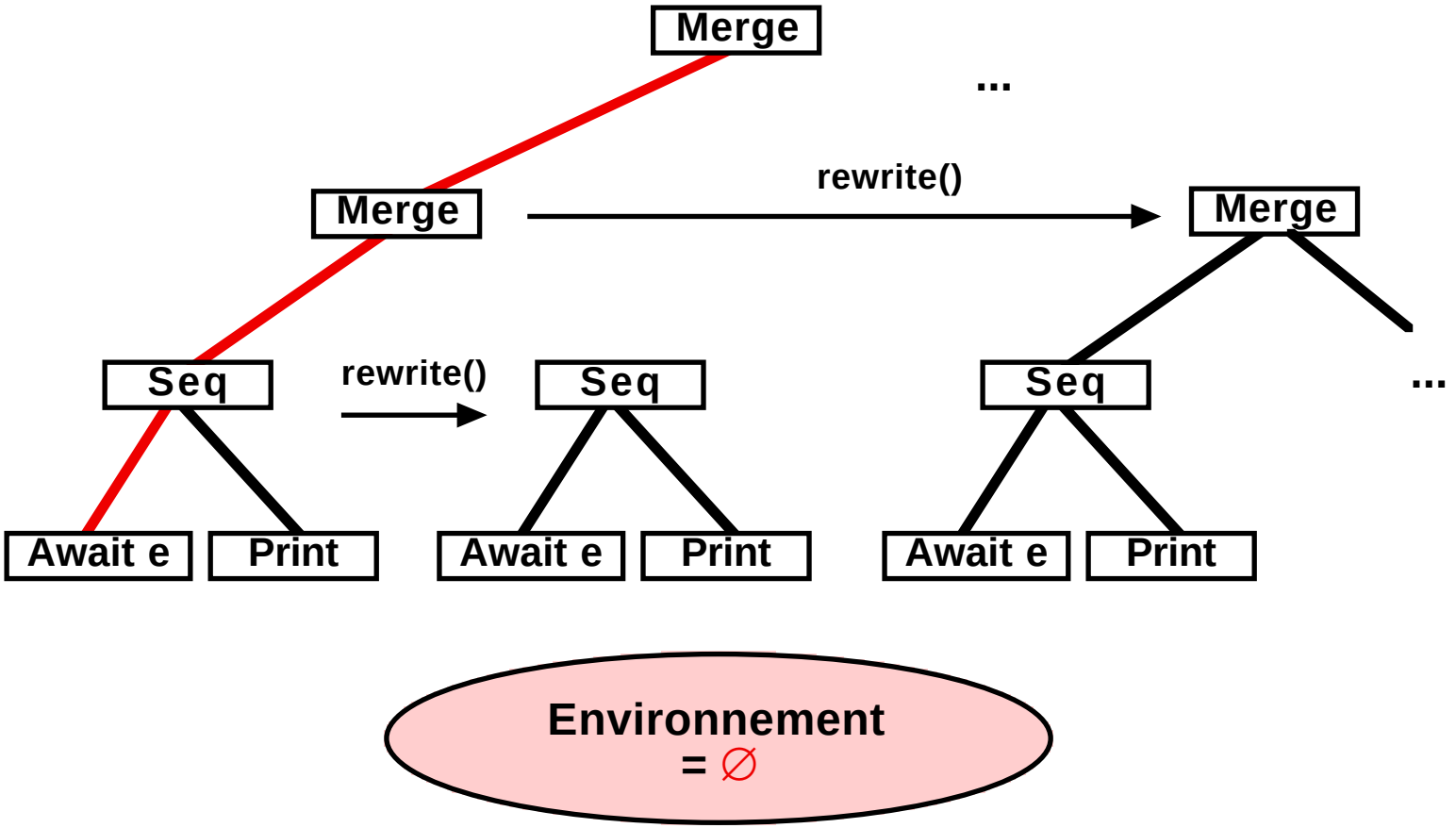
Rewrite-Replace



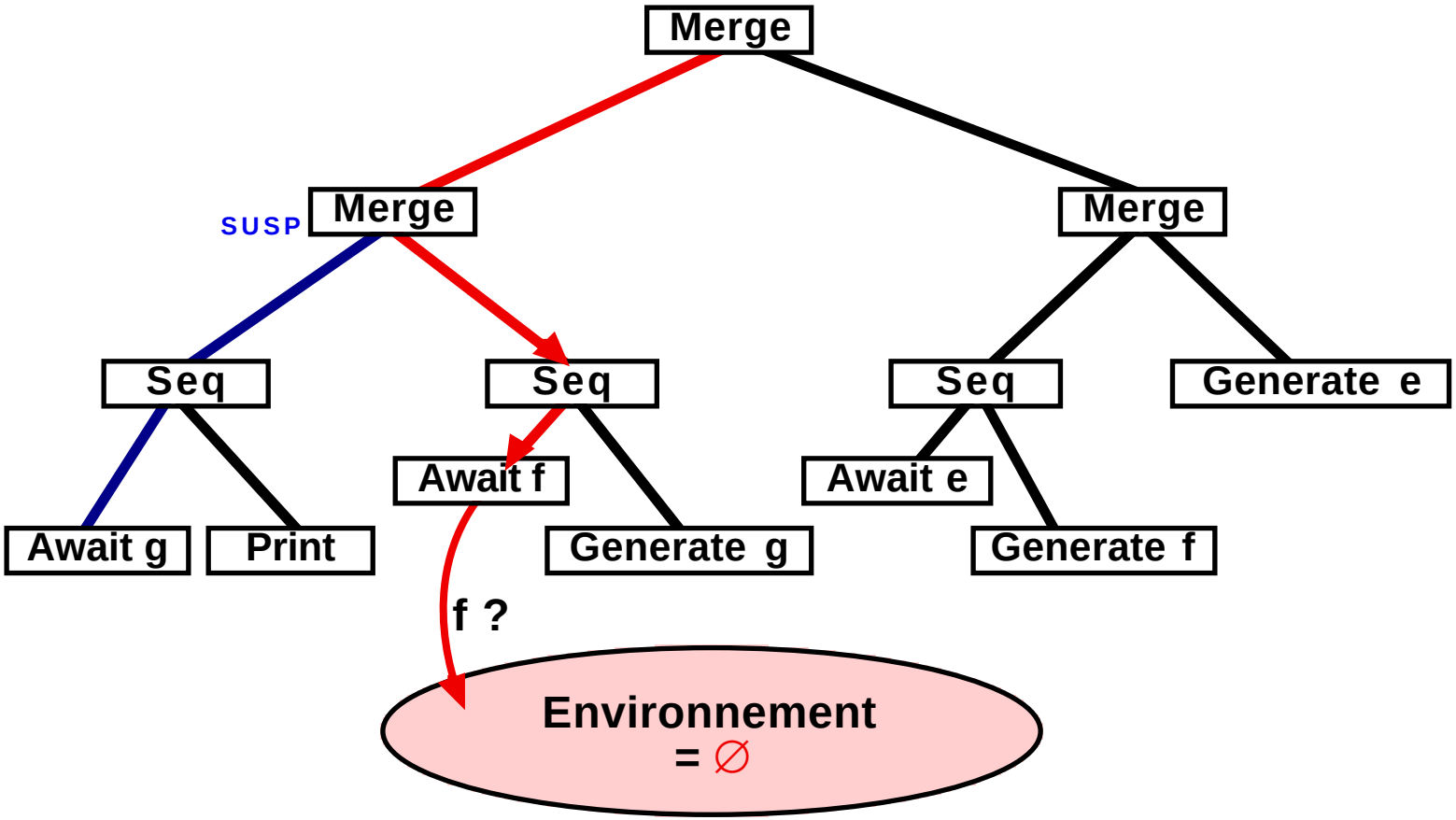
Rewrite-Replace



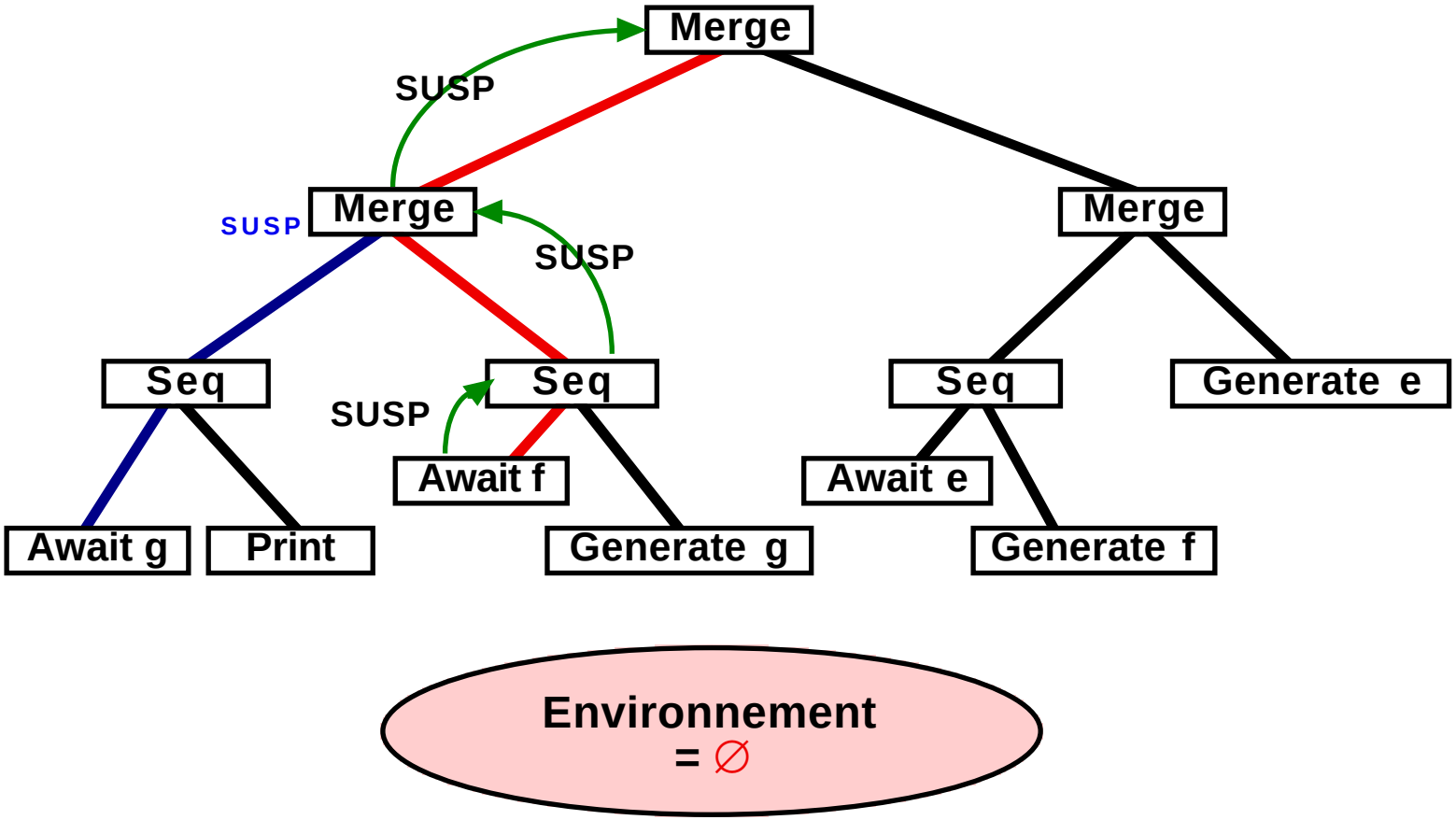
Rewrite


Merge
...
rewrite()
Merge
Merge
Seq
rewrite()
Seq
Seq
...
Await e
Print
Await e
Print
Await e
Print
Environnement
= ∅

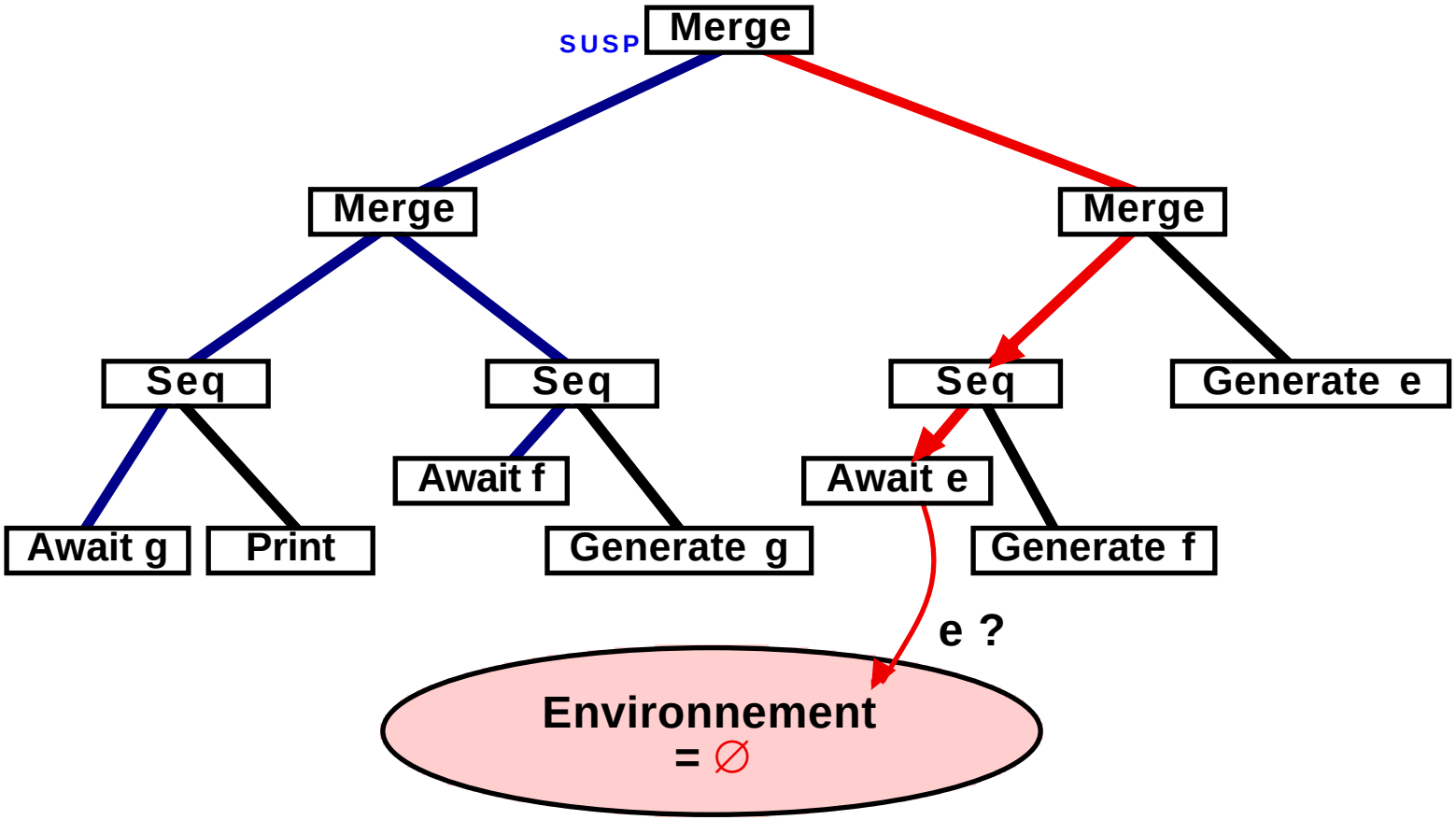
Replace


Merge
SUSP
Merge
Merge
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
f ?
Environnement
= ∅

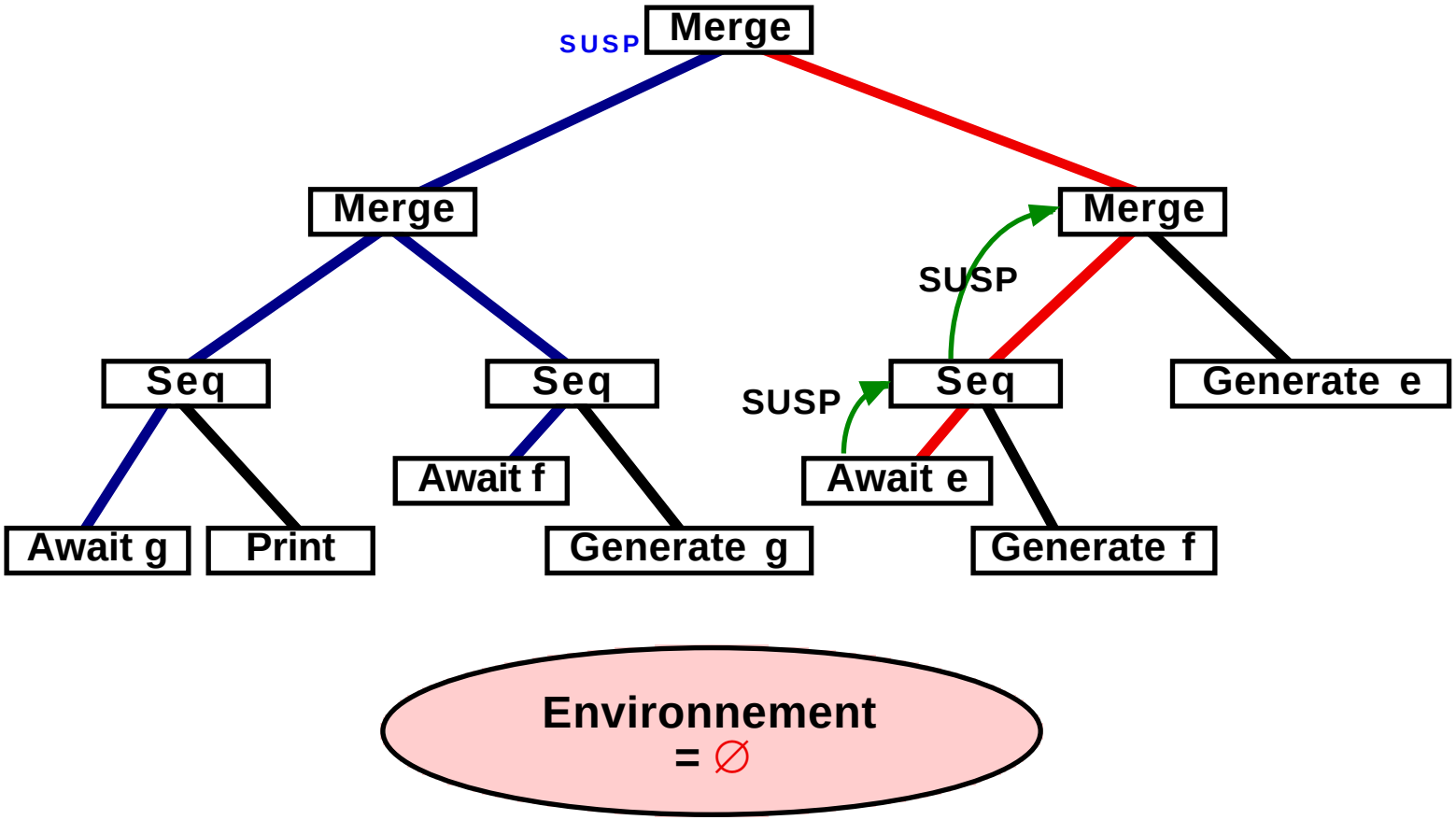
Replace


Merge
SUSP
SUSP
Merge
SUSP
Merge
Seq
SUSP
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
Environnement
= ∅

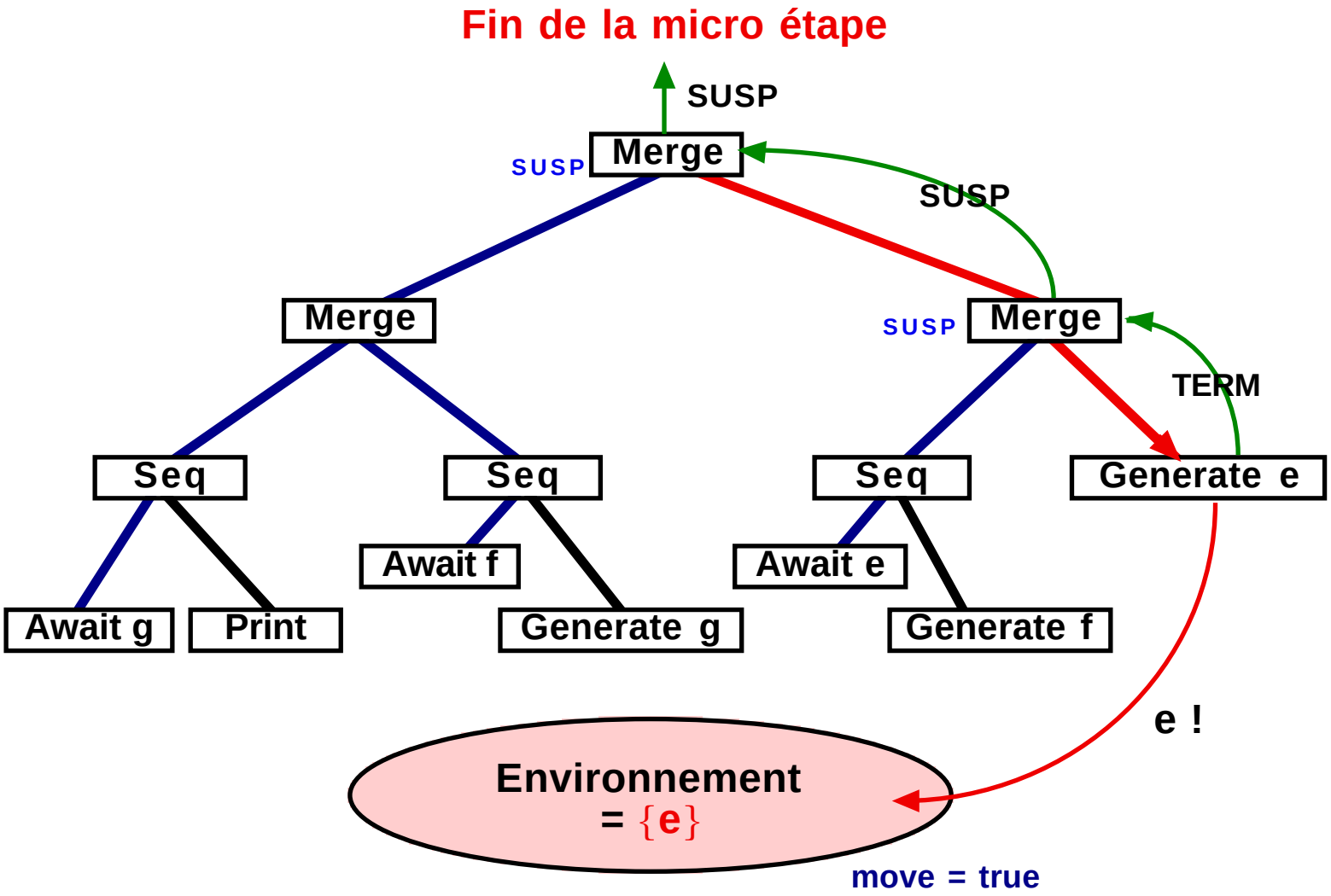
Replace


SUSP
Merge
Merge
Merge
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
e ?
Environnement
= ∅

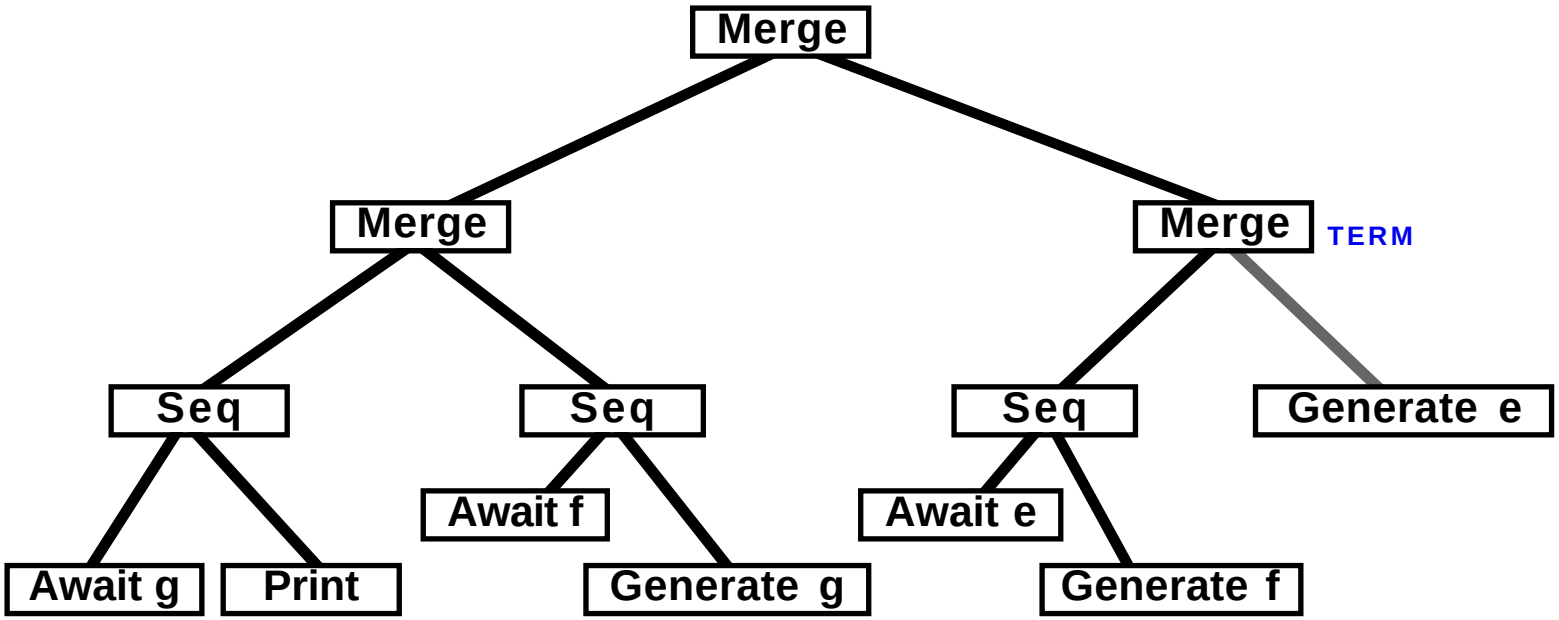
Replace


SUSP
Merge
Merge
Merge
SUSP
Seq
Seq
SUSP
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
Environnement
= ∅

Replace


Fin de la micro étape
SUSP
SUSP
Merge
SUSP
SUSP
Merge
Merge
TERM
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
e !
Environnement
= {e}
move = true

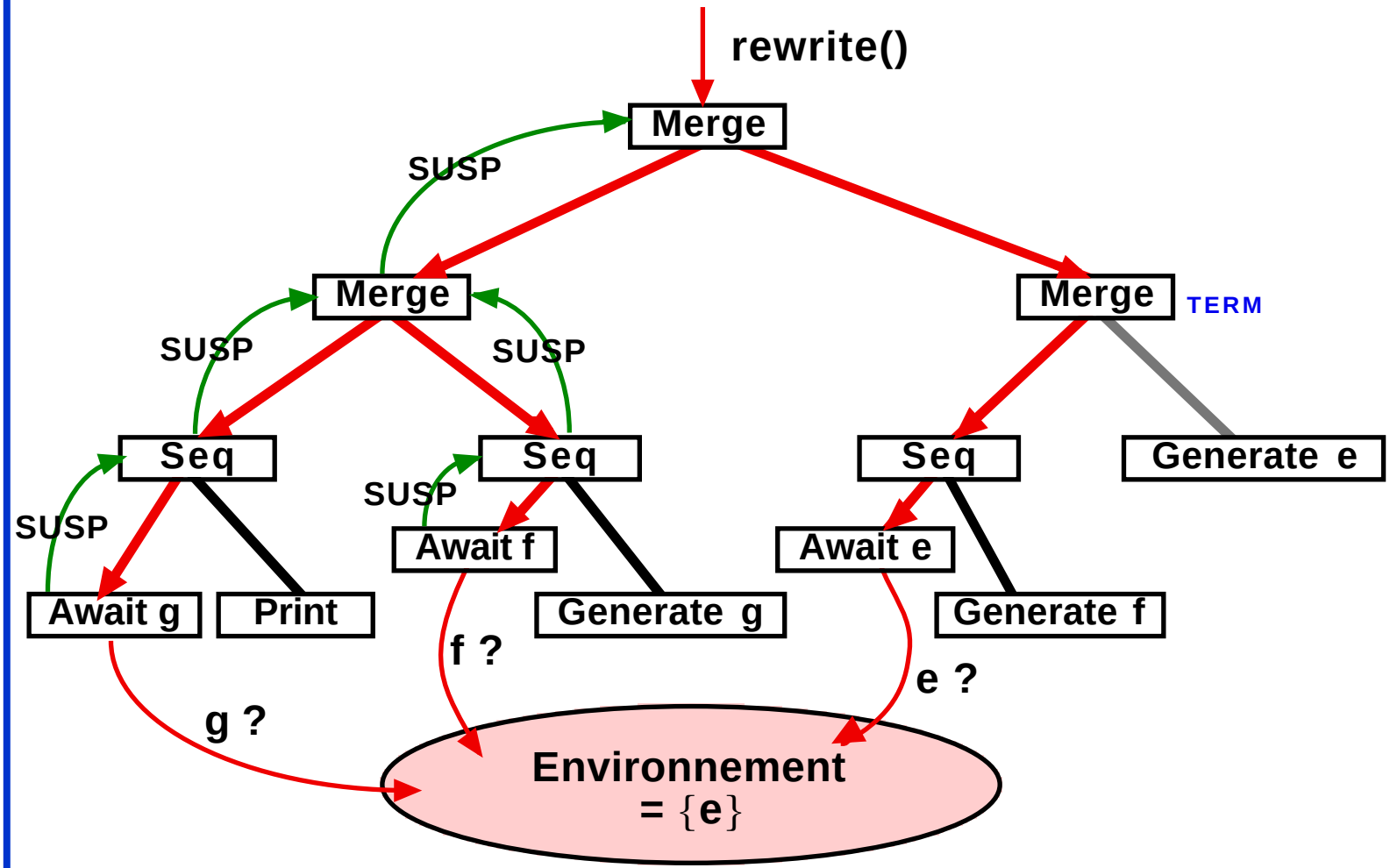
Replace



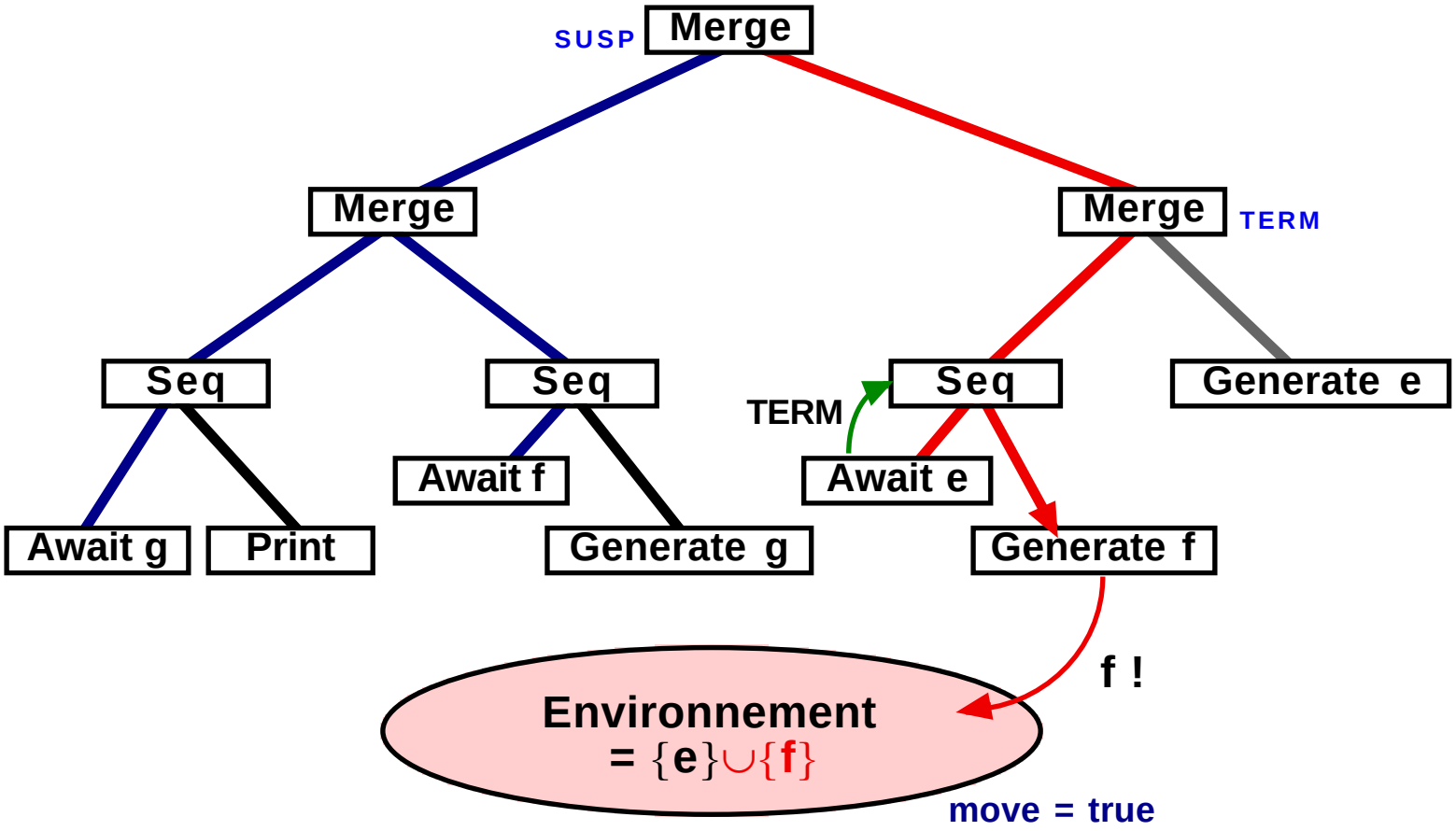
Environnement
= {e}

move = true

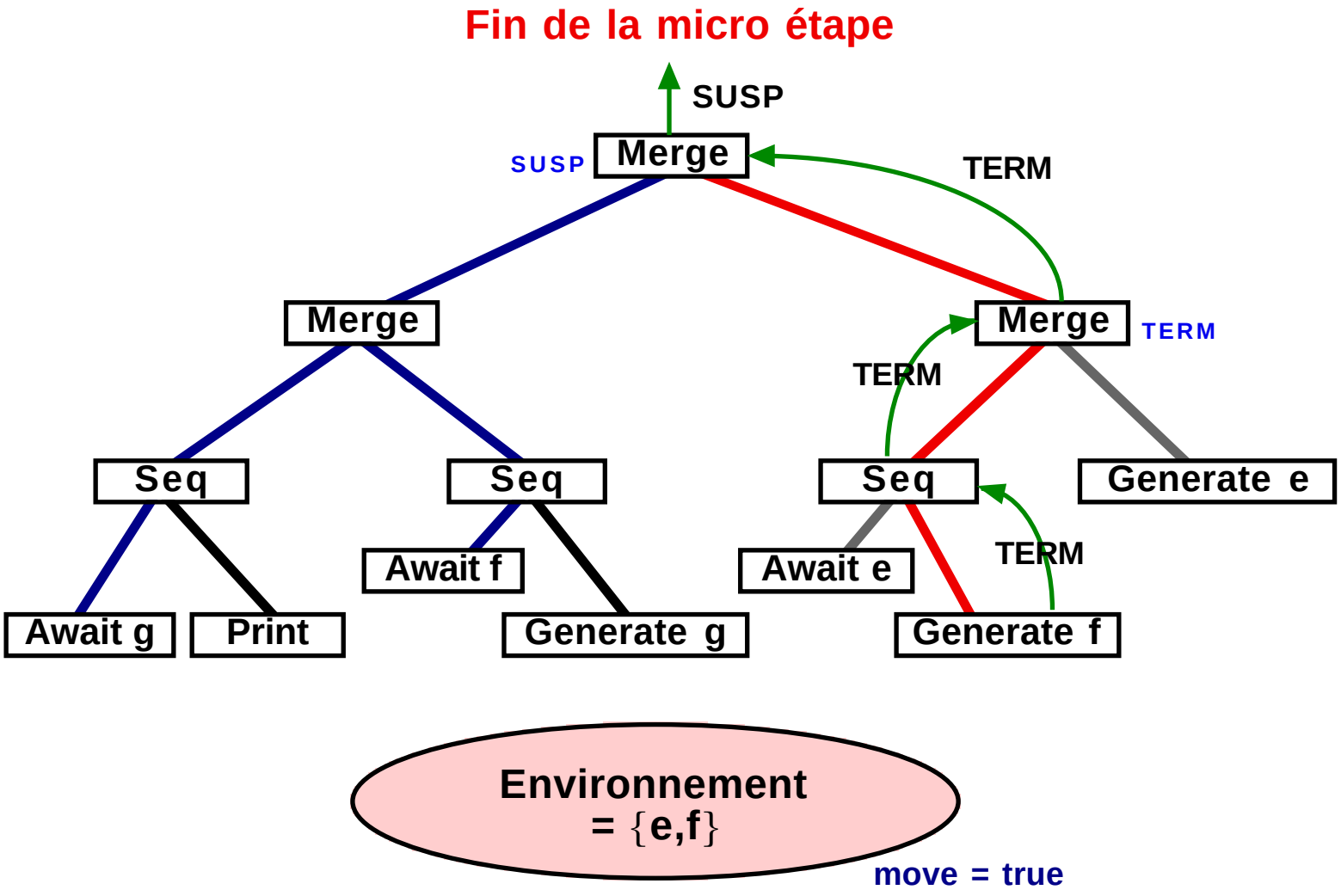
2ème micro étape

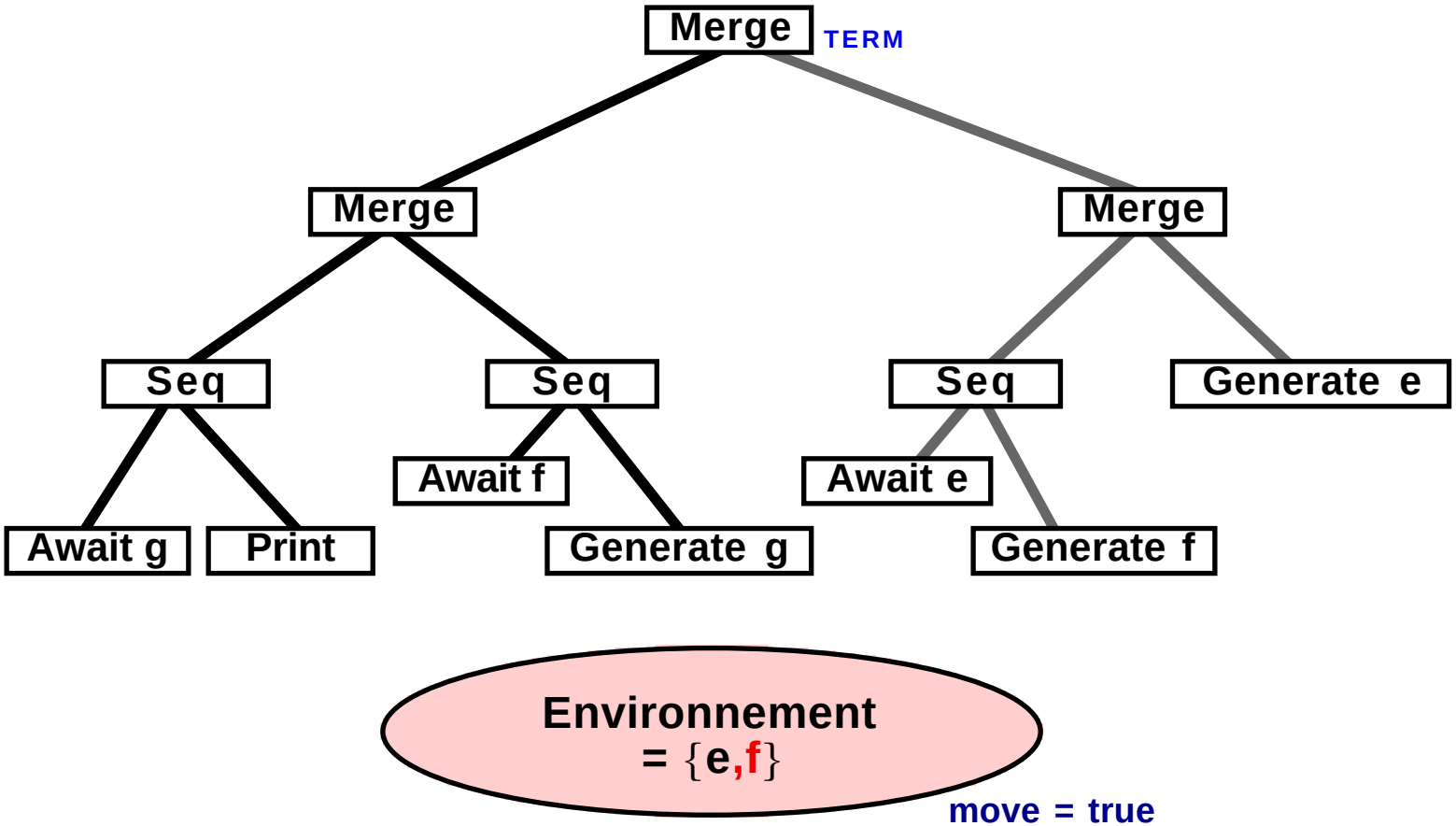


Replace

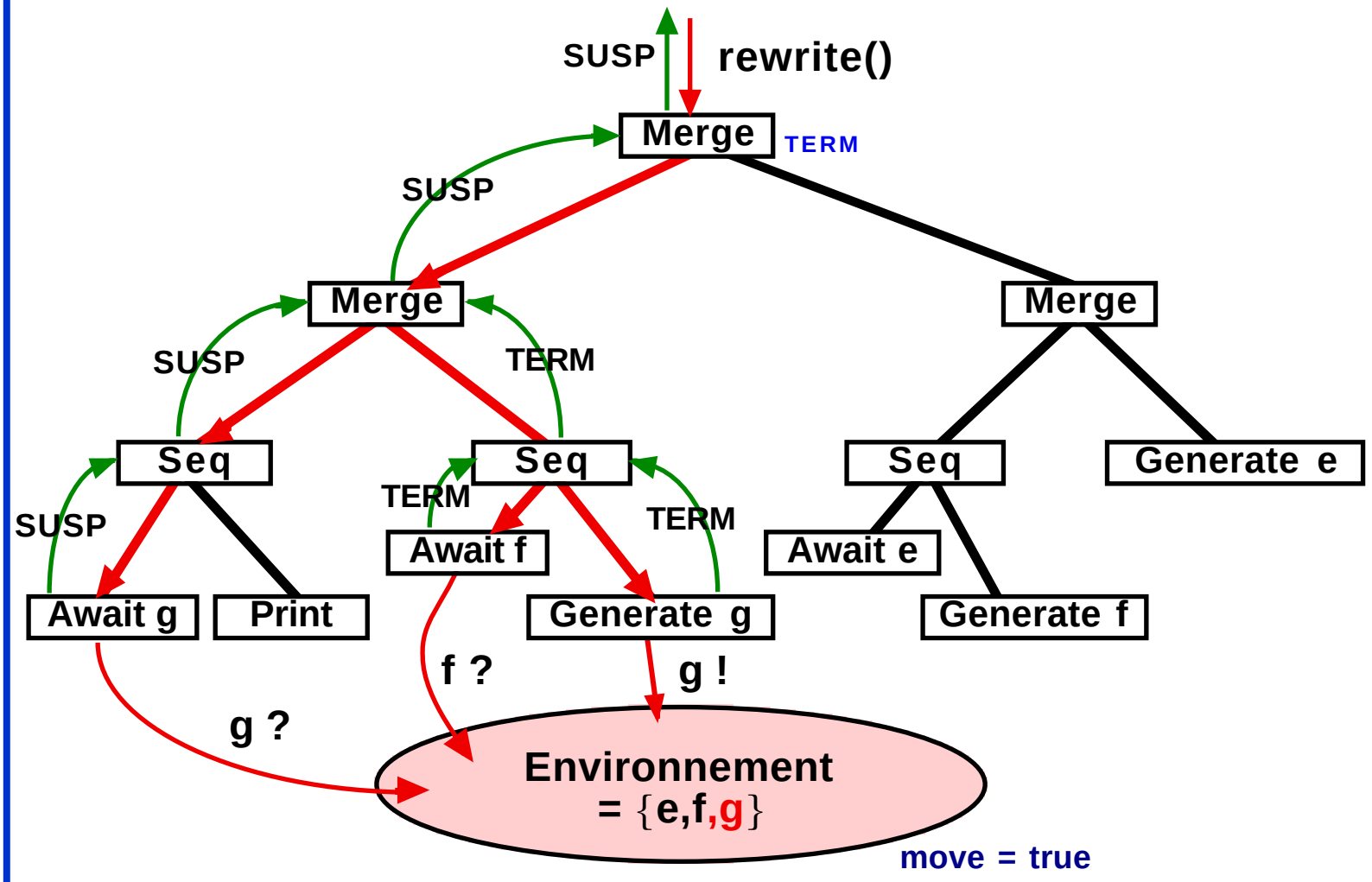

SUSP
Merge
Merge
Merge
TERM
Seq
Seq
Seq
Generate e
TERM
Await f
Await e
Await g
Print
Generate g
Generate f
f !
Environnement
= {e}∪{f}
move = true

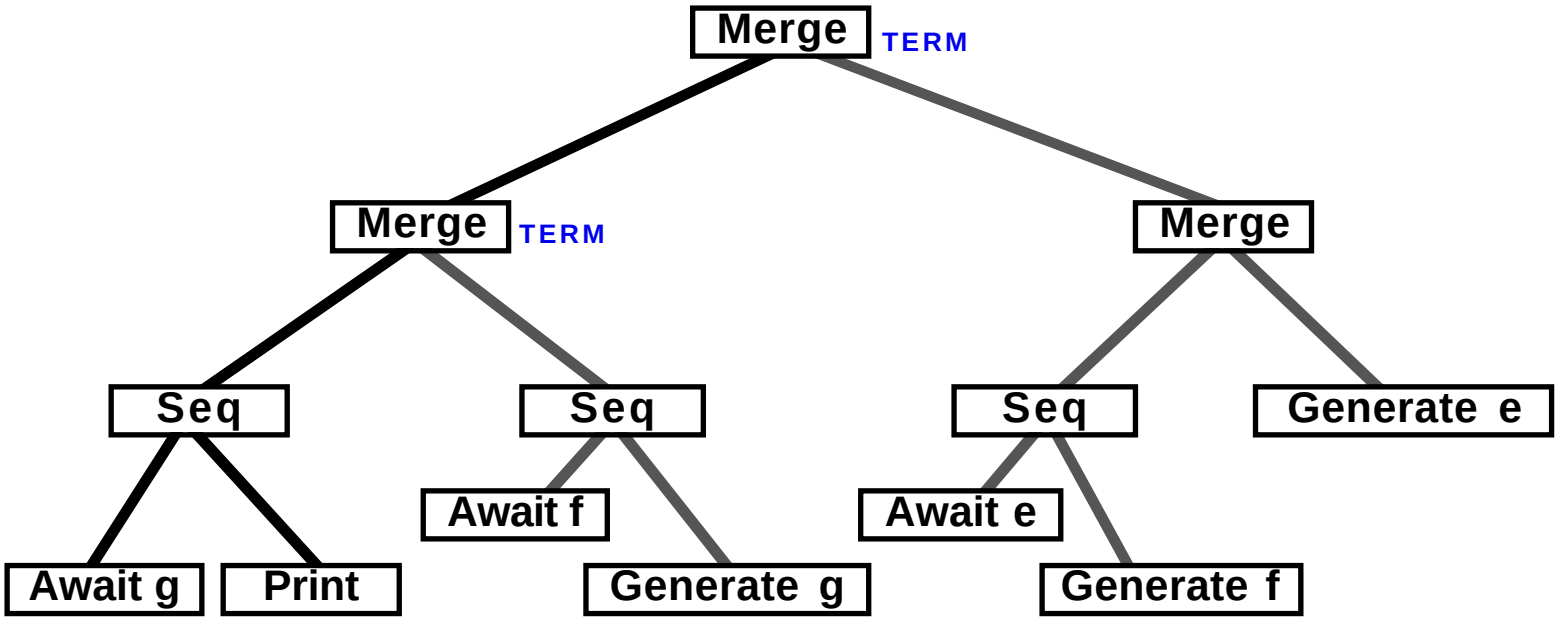
Replace


Fin de la micro étape
SUSP
SUSP
Merge
TERM
Merge
Merge
TERM
TERM
Seq
Seq
Seq
Generate e
Await f
Await e
TERM
Await g
Print
Generate g
Generate f
Environnement
= {e,f}
move = true


Merge
TERM
Merge
Merge
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
Environnement
= {e,f}
move = true

3ème micro étape

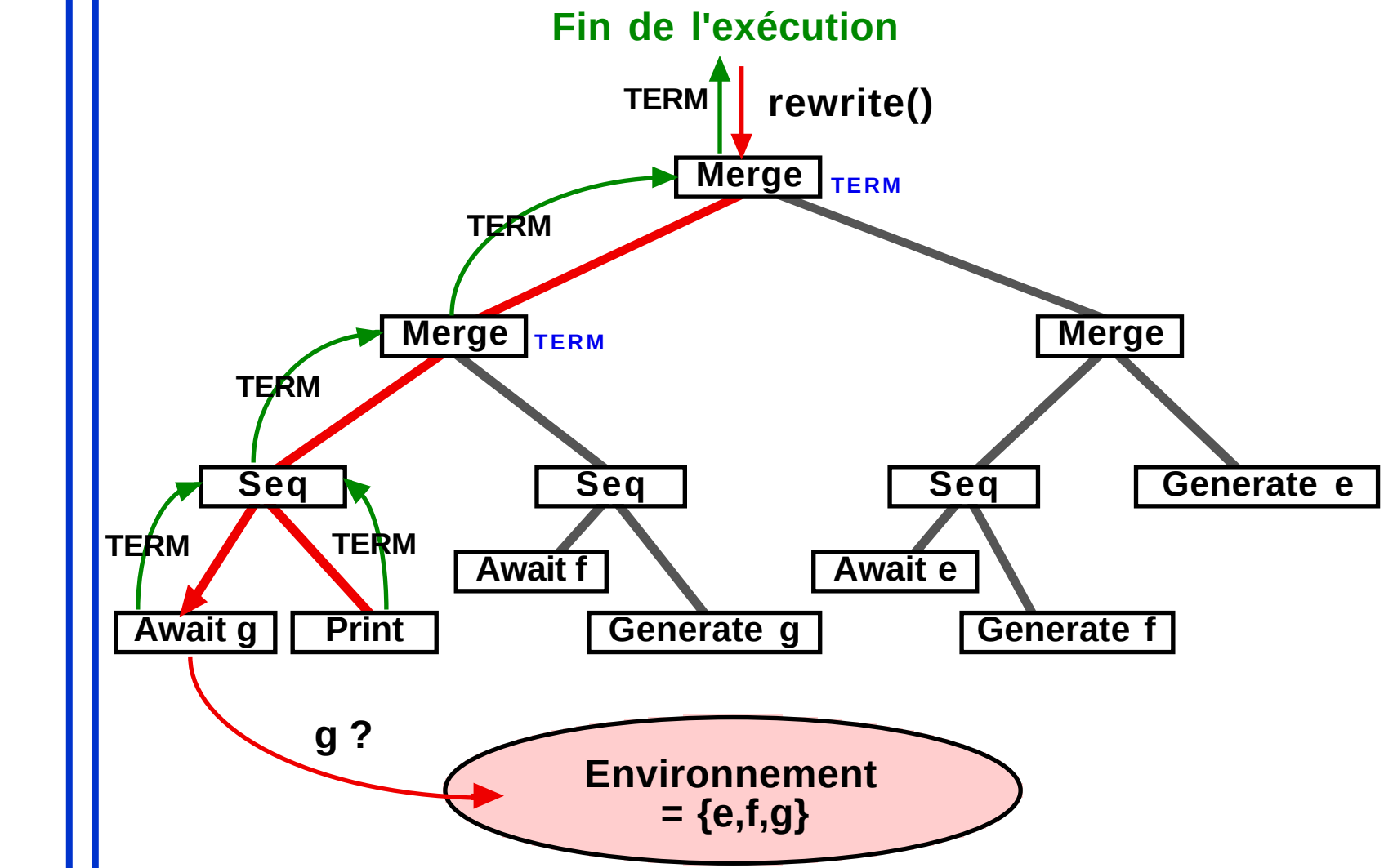




Environnement
= {e,f,g}

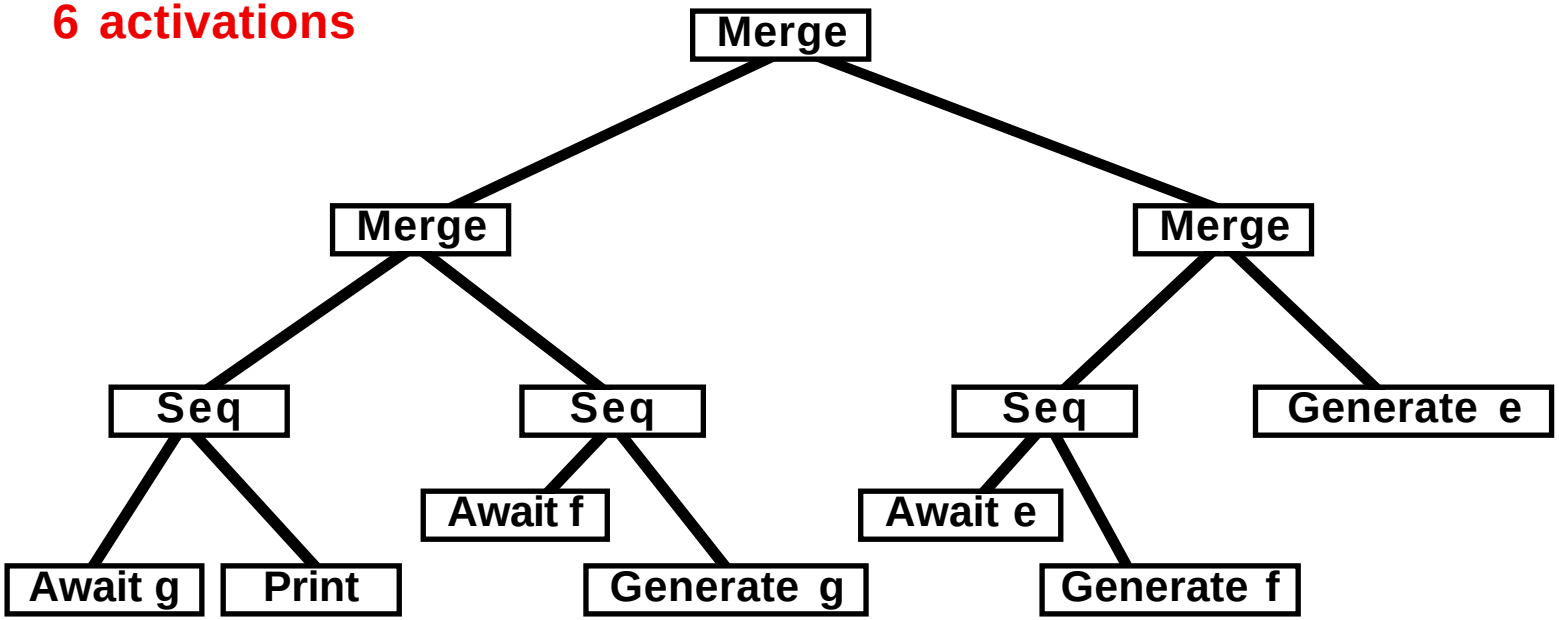
move = true

4ème micro étape



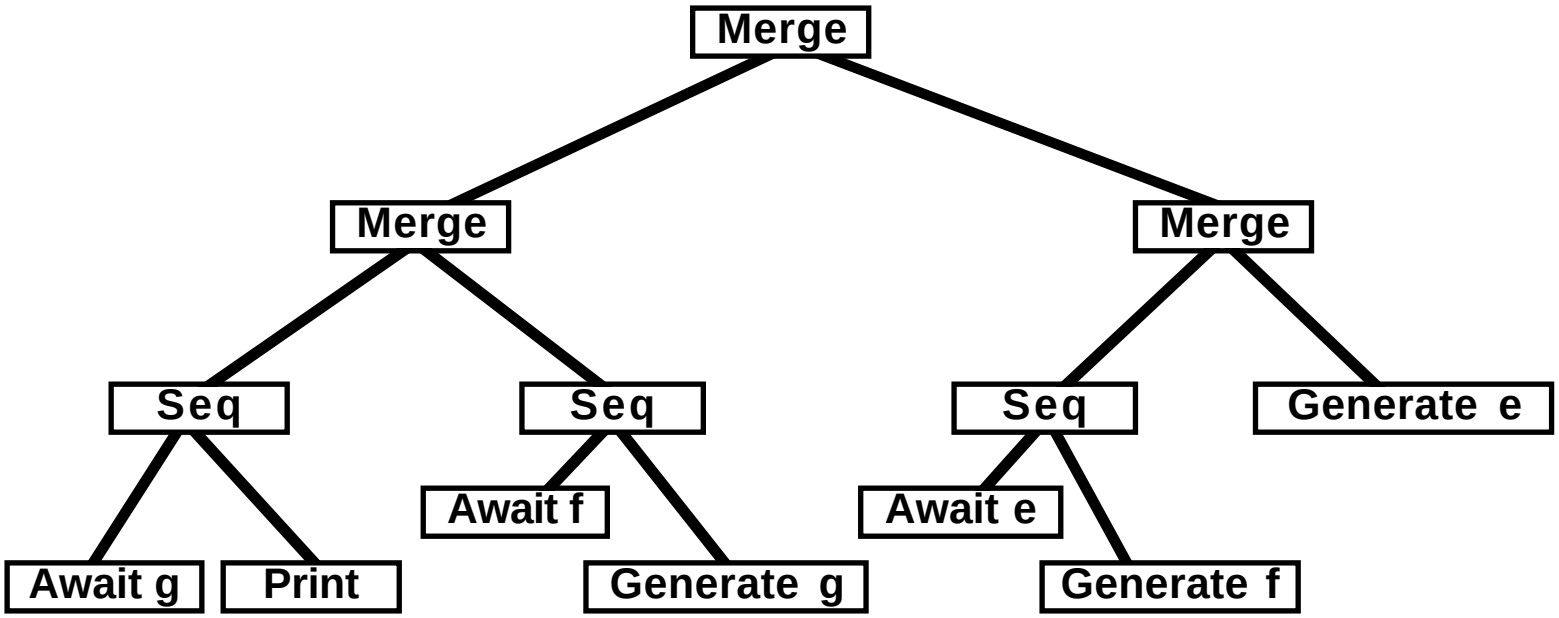
Bilan

4 micro étapes
6 activations



Complexité en : $\frac{n(n+1)}{2}$

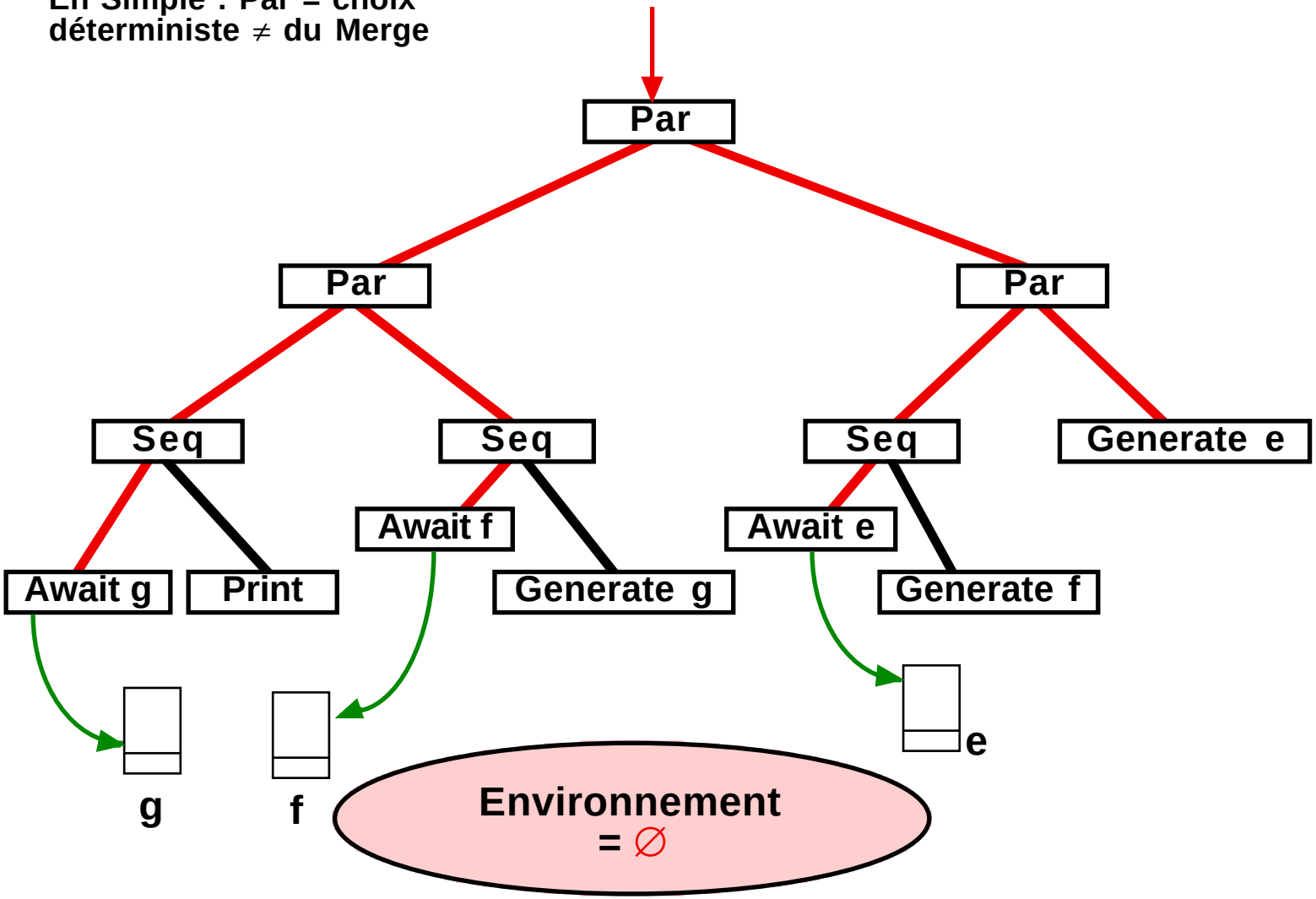
Ordonnancement contraire



Le programme s'exécute en une seule micro étape

Simple

En Simple : Par = choix déterministe ≠ du Merge

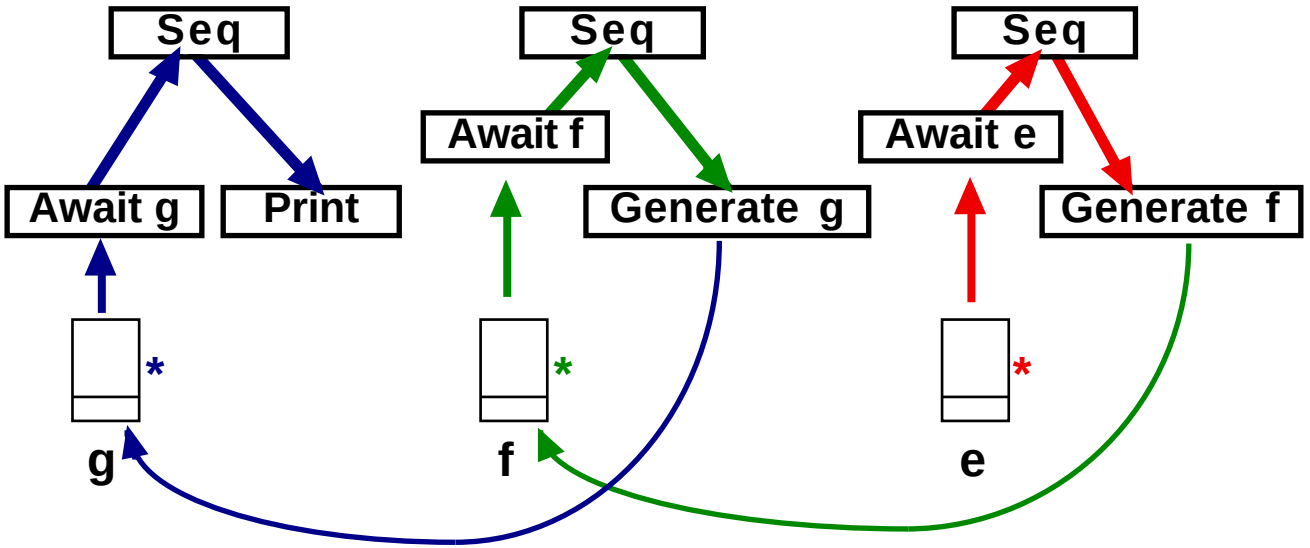


Simple

La structure du programme est éclatée dans des files d'attente.

Les opérateurs Par de plus haut niveau disparaissent

Croissance linéaire avec le nombre de composants et d'événements à traiter effectivement



Simple

Simple réduit la complexité inter-instant : évite la diffusion de fin d'instant inutile

Ex : Await("e") si e n'est pas là

File d'attente des composants stoppés

Pbs

algorithmique plus complexe (abonnement désabonnement dans des files d'attente)

2 sens d'exécution :

- **descendante (ex : pour la toute première activation)**
- **remontante (activation par les fils d'attentes)**

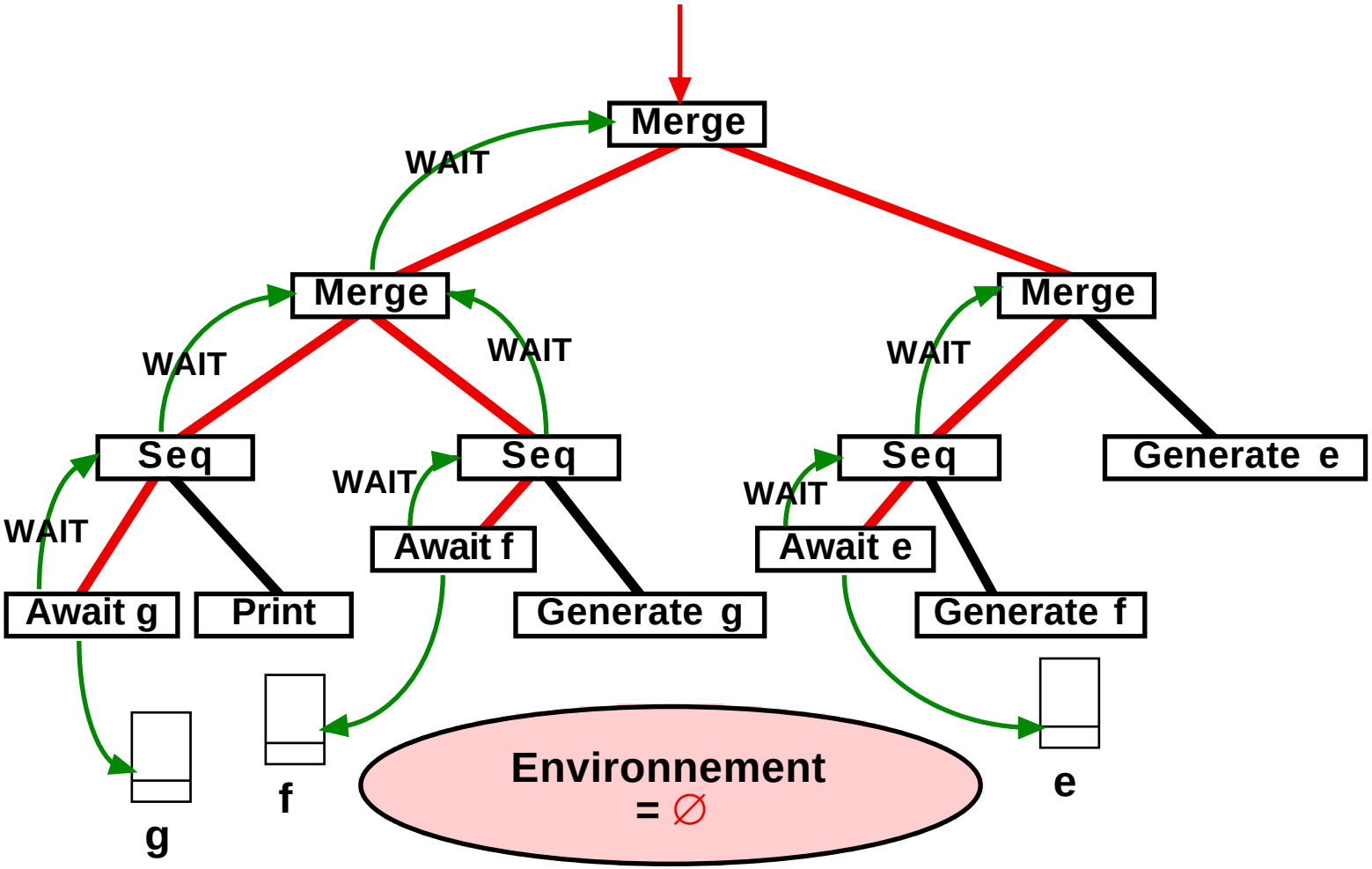
Exécution non structurelle => pas de règles SOS

Mise au point et analyse délicate.

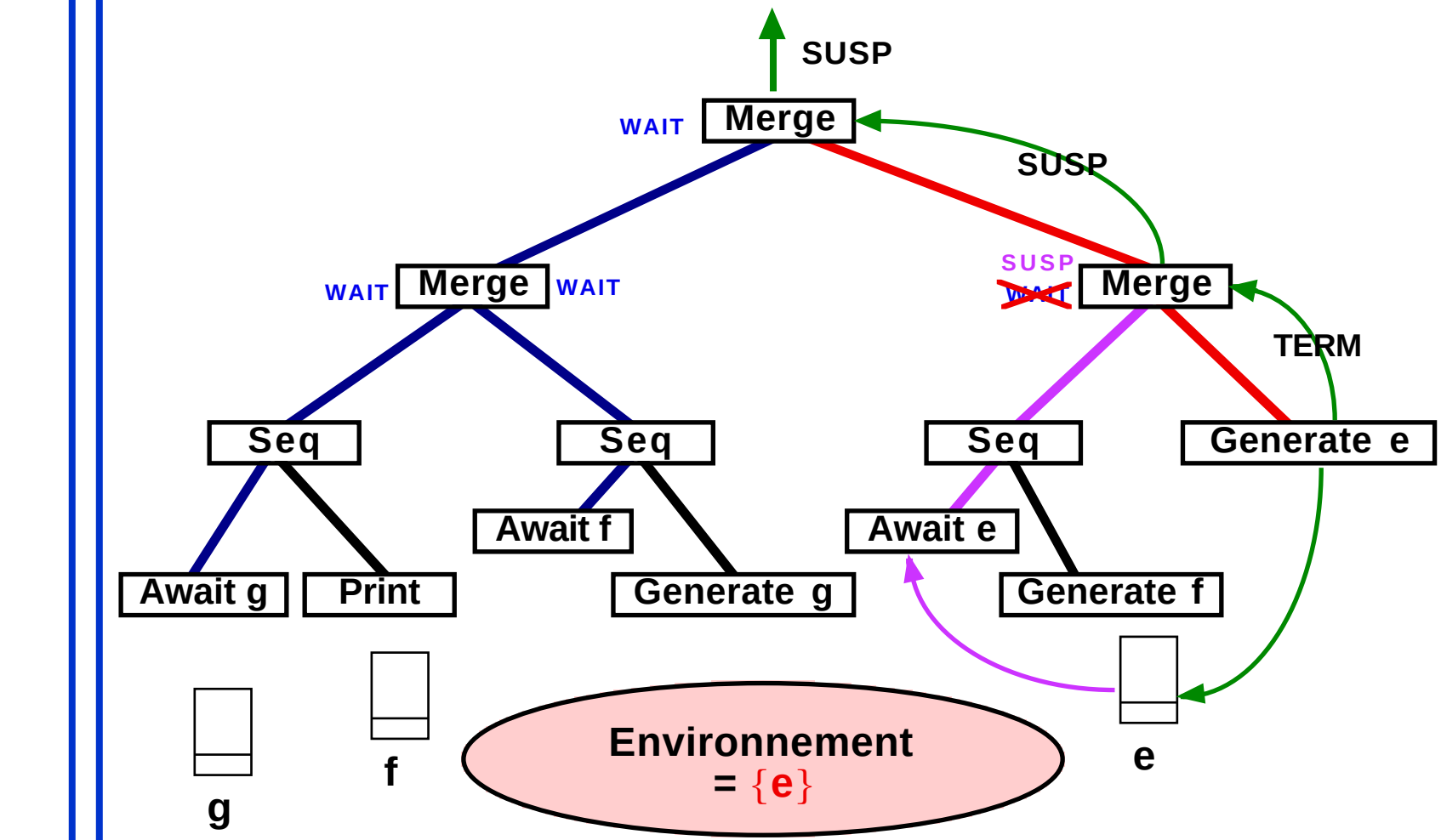
Storm

- Etend Replace
- Différencie attente sur événement et suspension
- Mécanisme de files d'attentes
- Exécution uniquement structurelle
- Mécanisme de précurseurs

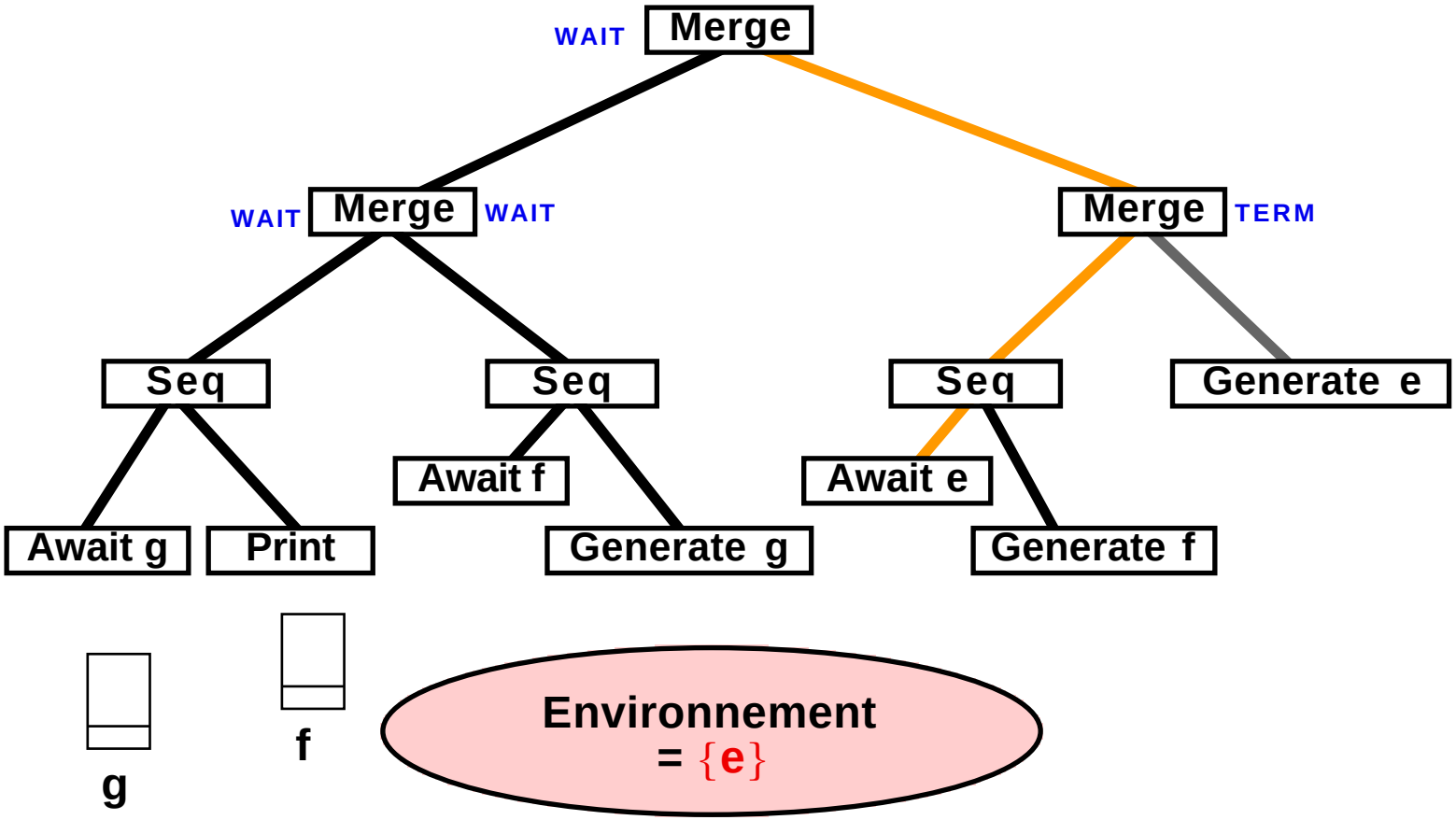
Storm


Merge
WAIT
Merge
Merge
WAIT
WAIT
WAIT
Seq
Seq
Seq
Generate e
WAIT
WAIT
WAIT
Await f
Await e
Await g
Print
Generate g
Generate f
g
f
e
Environnement
= ∅

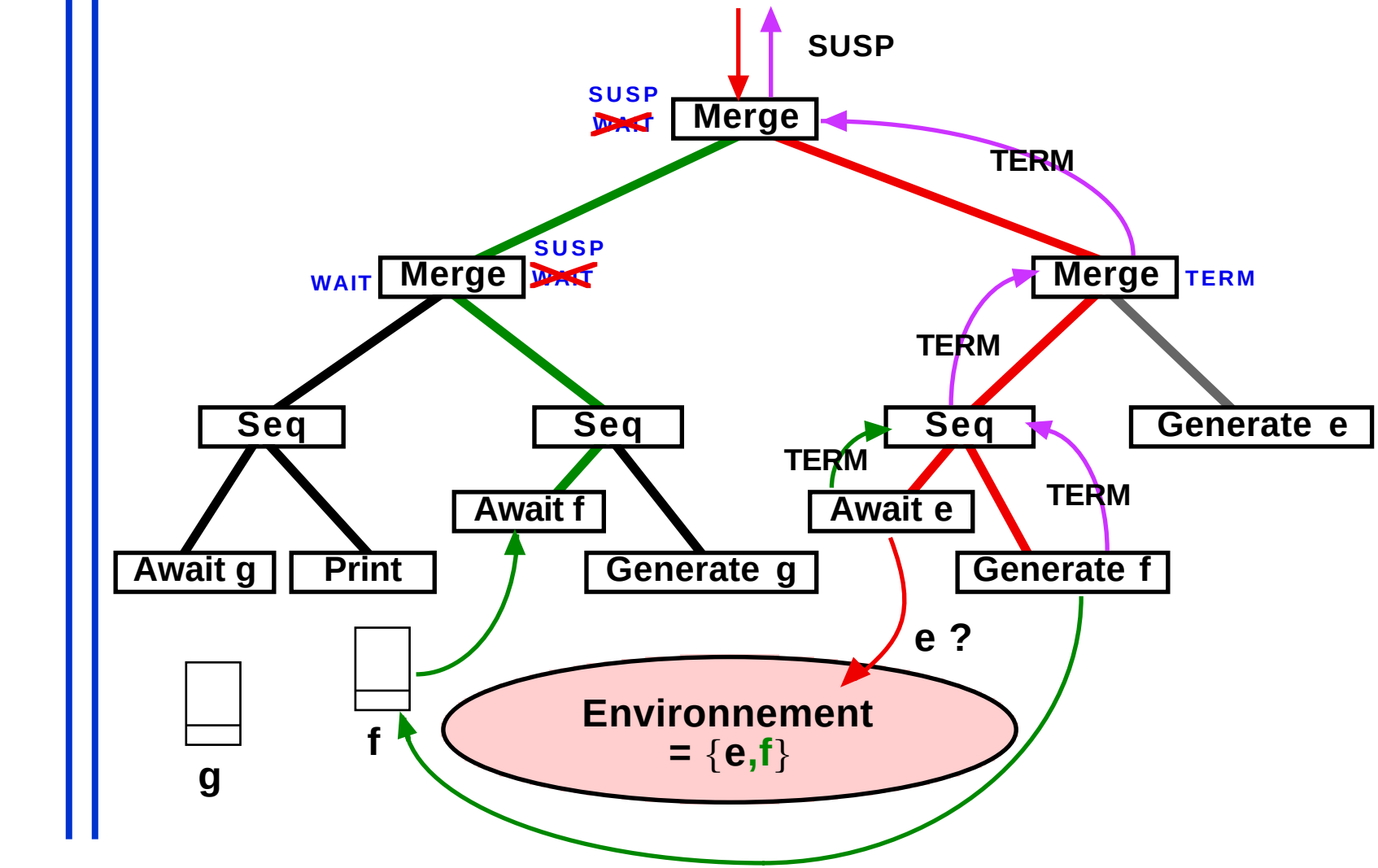
Storm


SUSP
WAIT
Merge
SUSP
WAIT
Merge
WAIT
SUSP
Merge
TERM
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
g
f
e
Environnement
= {e}

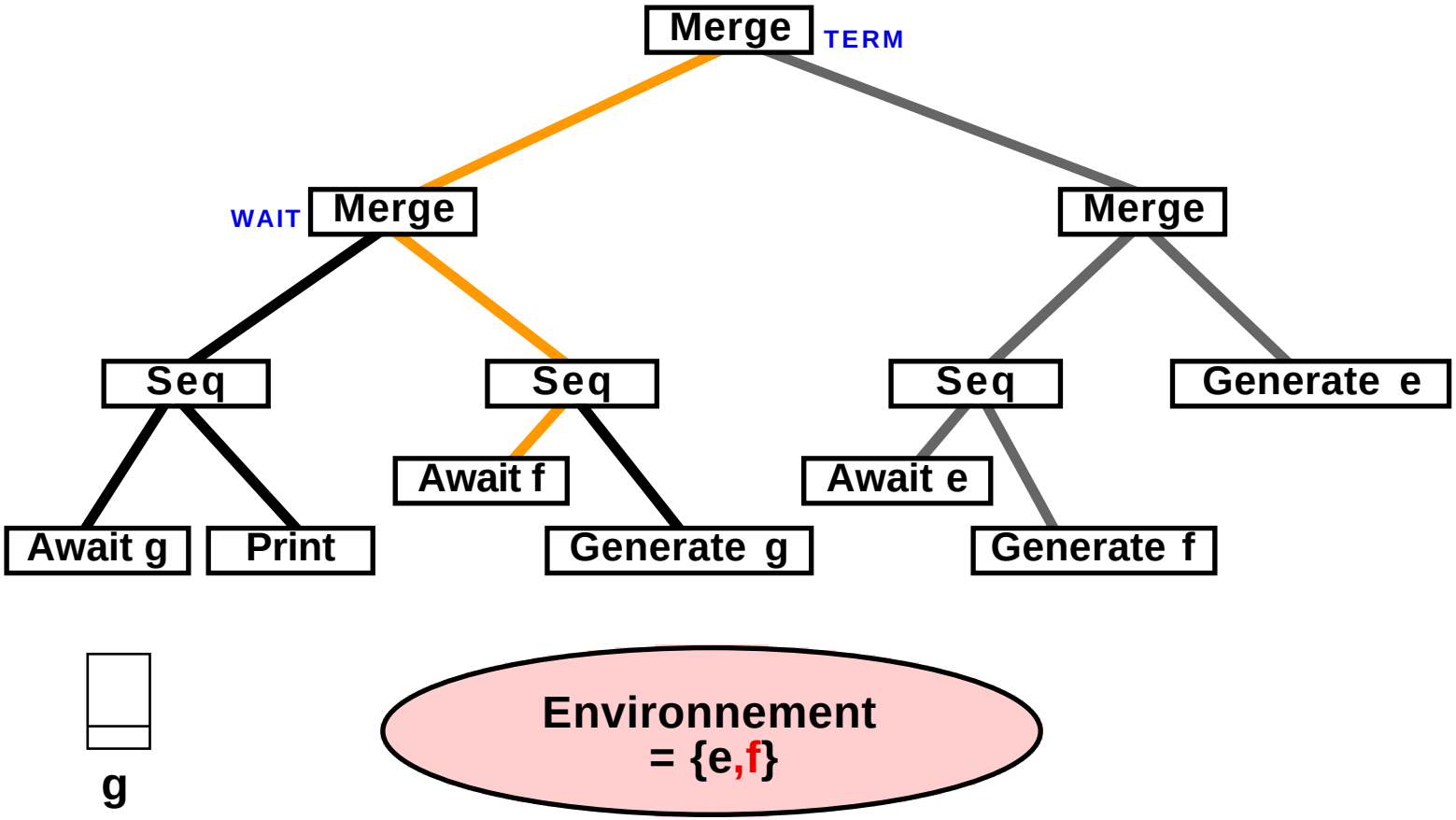
Storm


WAIT
Merge
WAIT
Merge
WAIT
Merge
TERM
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
g
f
Environnement
= {e}

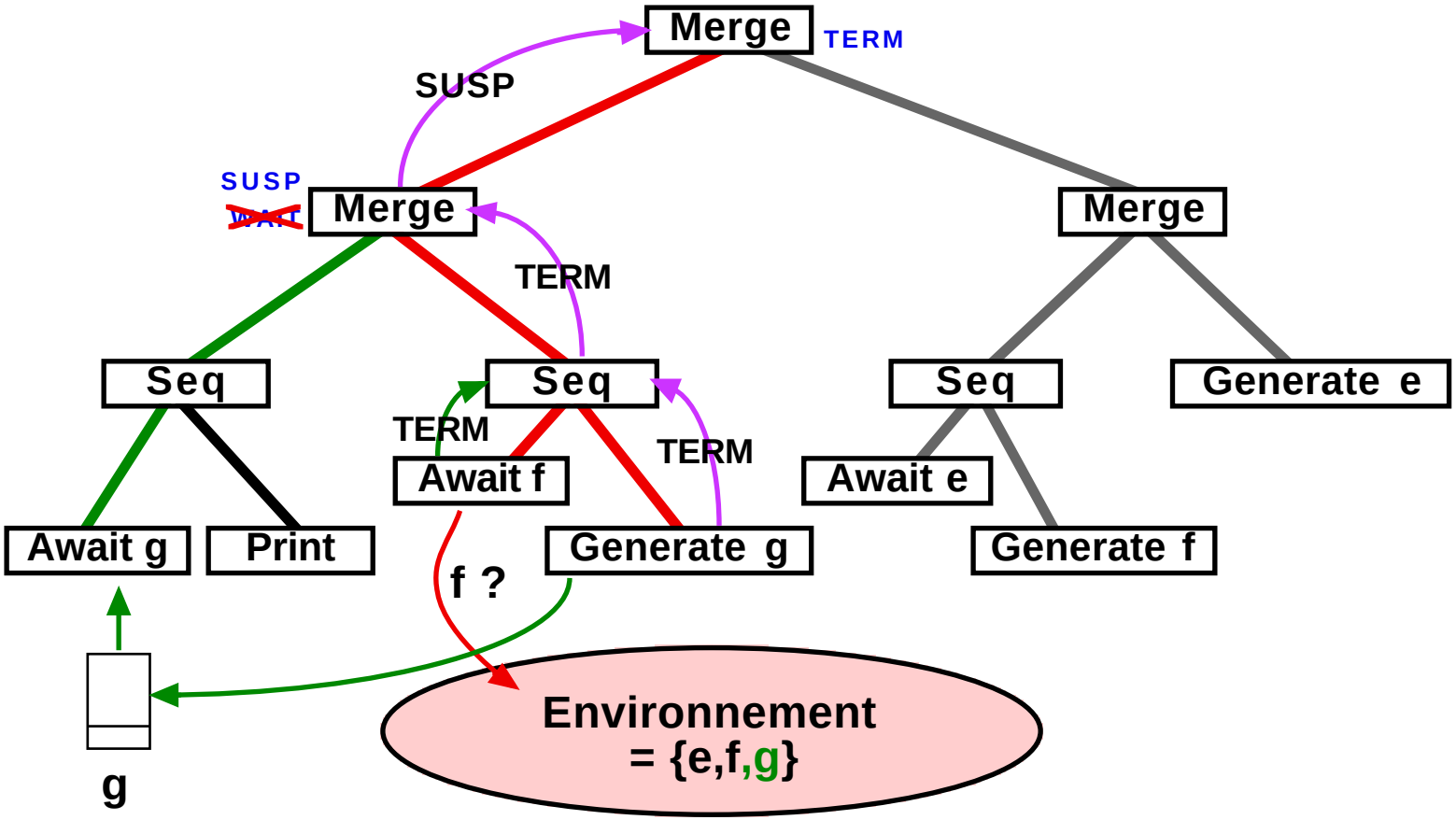
Storm


SUSP
SUSP
WAIT
Merge
TERM
WAIT
Merge
SUSP
WAIT
Merge
TERM
TERM
Seq
Seq
Seq
Generate e
TERM
TERM
Await f
Await e
Await g
Print
Generate g
Generate f
e ?
Environnement
= {e,f}
f
g

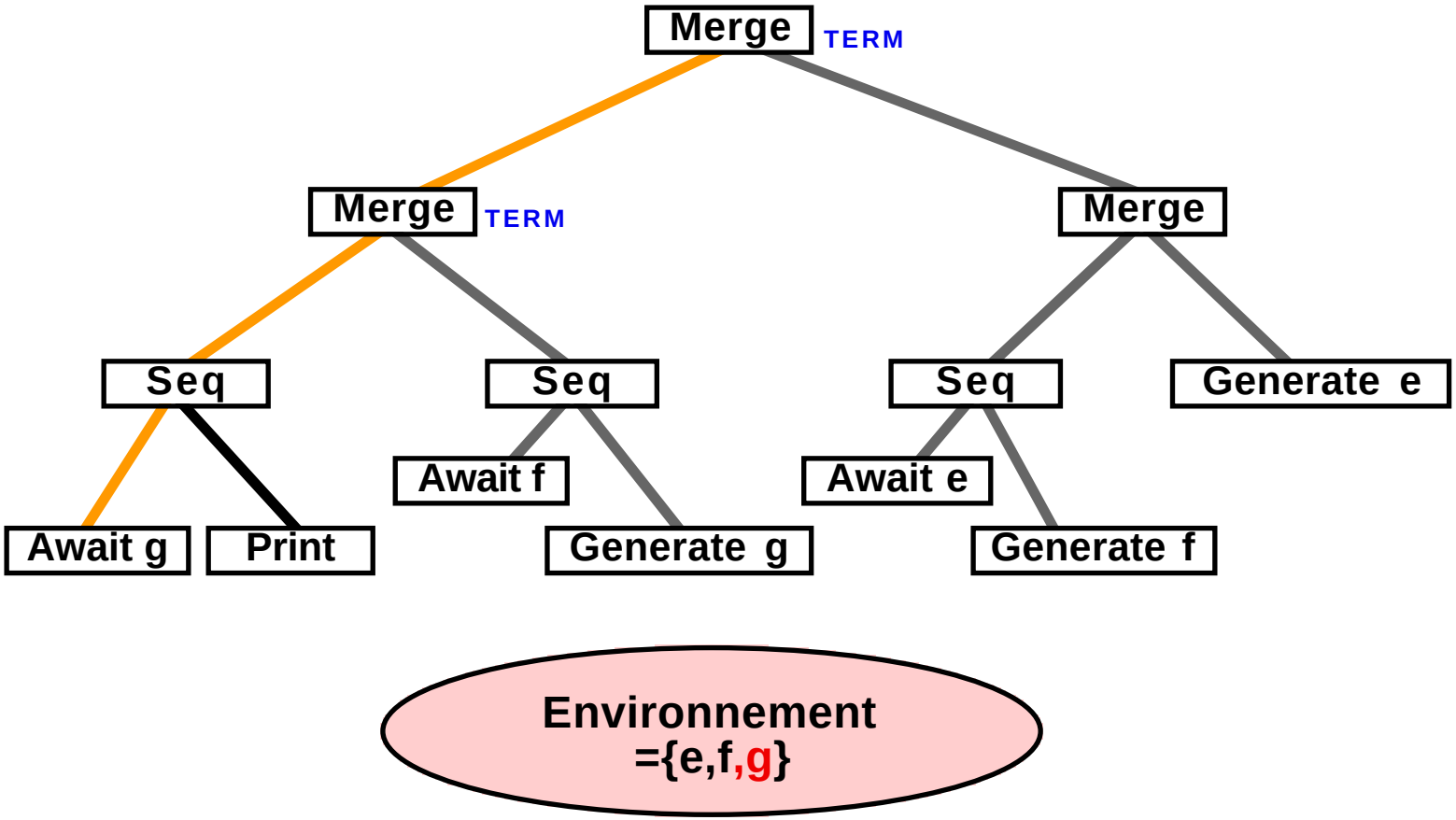
Storm


Merge
TERM
WAIT
Merge
Merge
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
g
Environnement
= {e,f}

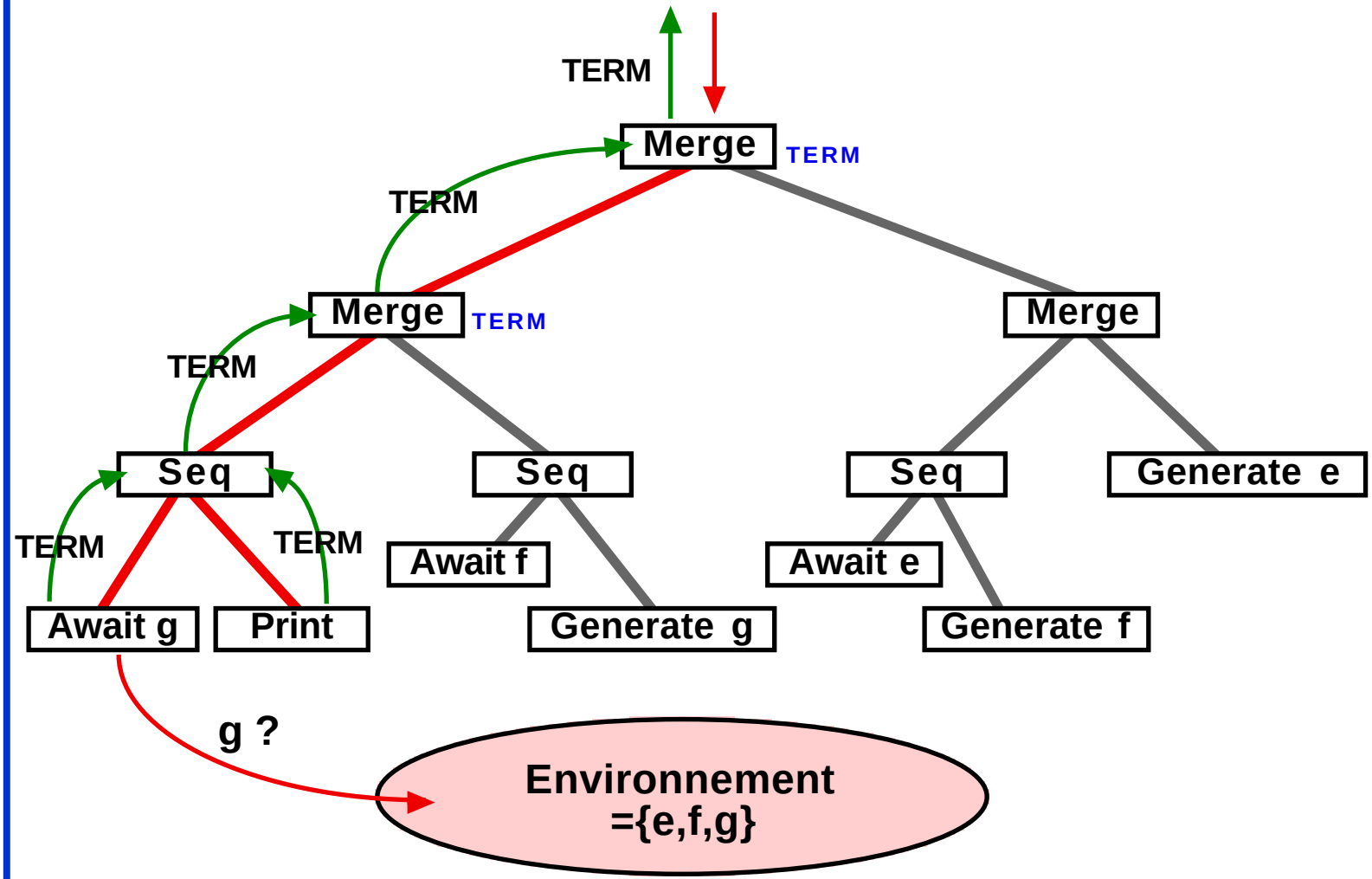
Storm


Merge
TERM
SUSP
SUSP
WAIT
Merge
Merge
TERM
Seq
Seq
Seq
Generate e
TERM
TERM
Await f
Await e
Await g
Print
Generate g
Generate f
f ?
Environnement
= {e,f,g}
g

Storm


Merge
TERM
Merge
TERM
Merge
Seq
Seq
Seq
Generate e
Await f
Await e
Await g
Print
Generate g
Generate f
Environnement
={e,f,g}

Storm


TERM
Merge
TERM
TERM
Merge
TERM
Merge
TERM
Seq
Seq
Seq
Generate e
TERM
TERM
Await f
Await e
Await g
Print
Generate g
Generate f
g ?
Environnement
={e,f,g}

Storm

- **Réduit la complexité due aux *cascades inverses***
- **Exécution uniquement descendante**
- **Sémantique dérivée de Replace**
- **Formalisation des précurseurs par un second système de règles**
- **Ne traite pas la complexité inter-instants.**
- **Performances dépendantes de la structure du programme**

Implémentation

```
abstract public class Instruction implements ...
{
    public EnvironmentImpl env;
    public Instruction parent;
    public void bind(EnvironmentImpl env,Instruction aParent){
        this.env = env; parent = aParent;
    }
    public void setParent(Instruction aParent){ parent = aParent; }

    abstract public byte rewrite();
    abstract public void reset();

...
    public void zap(Instruction from){
        throw new InternalError("Not zappable Instruction");
    }
}
```

instruction parente dans l'arbre du programme

exécution structurelle selon les règles de réécriture

implémentation des précurseurs

par défaut une instruction ne peut pas être le point de départ d'un

Implémentation

```
public class Par extends BinaryInstruction
{
  public byte leftFlag = SUSP, rightFlag = SUSP;
  public Par(Program left,Program right){ super(left,right); }
  public void reset(){ super.reset(); leftFlag = rightFlag = SUSP; }
...
  public byte rewrite(){
      if((SUSP == leftFlag)||((WAIT == leftFlag)&&env.eoi)){
          leftFlag = left.rewrite();}
      if((SUSP == rightFlag)||((WAIT == rightFlag)&&env.eoi)){
          rightFlag = right.rewrite();}
      if((TERM == leftFlag)&&(TERM == rightFlag)){ return TERM; }
      if((SUSP == leftFlag)||(SUSP == rightFlag)){ return SUSP; }
      if((WAIT == leftFlag)||(WAIT == rightFlag)){ return WAIT; }
      if(STOP == rightFlag){ rightFlag = SUSP; }
      if(STOP == leftFlag){ leftFlag = SUSP; }
      return STOP;
  }
...
  public void zap(Instruction from){
      if(from == left){
          if(WAIT == leftFlag){
              leftFlag = SUSP; if(SUSP != rightFlag){ parent.zap(this); }
          }
      }
      else if(from == right){
          if(WAIT == rightFlag){
              rightFlag = SUSP; if(SUSP != leftFlag){ parent.zap(this); }
          }
      }
      else{ throw new InternalError("Awaked by unknown son"); }
  }
}
```

	SUSP	WAIT	STOP	TERM
SUSP	SUSP	SUSP	SUSP	SUSP
WAIT	SUSP	WAIT	WAIT	WAIT
STOP	SUSP	WAIT	STOP	STOP
TERM	SUSP	WAIT	STOP	TERM

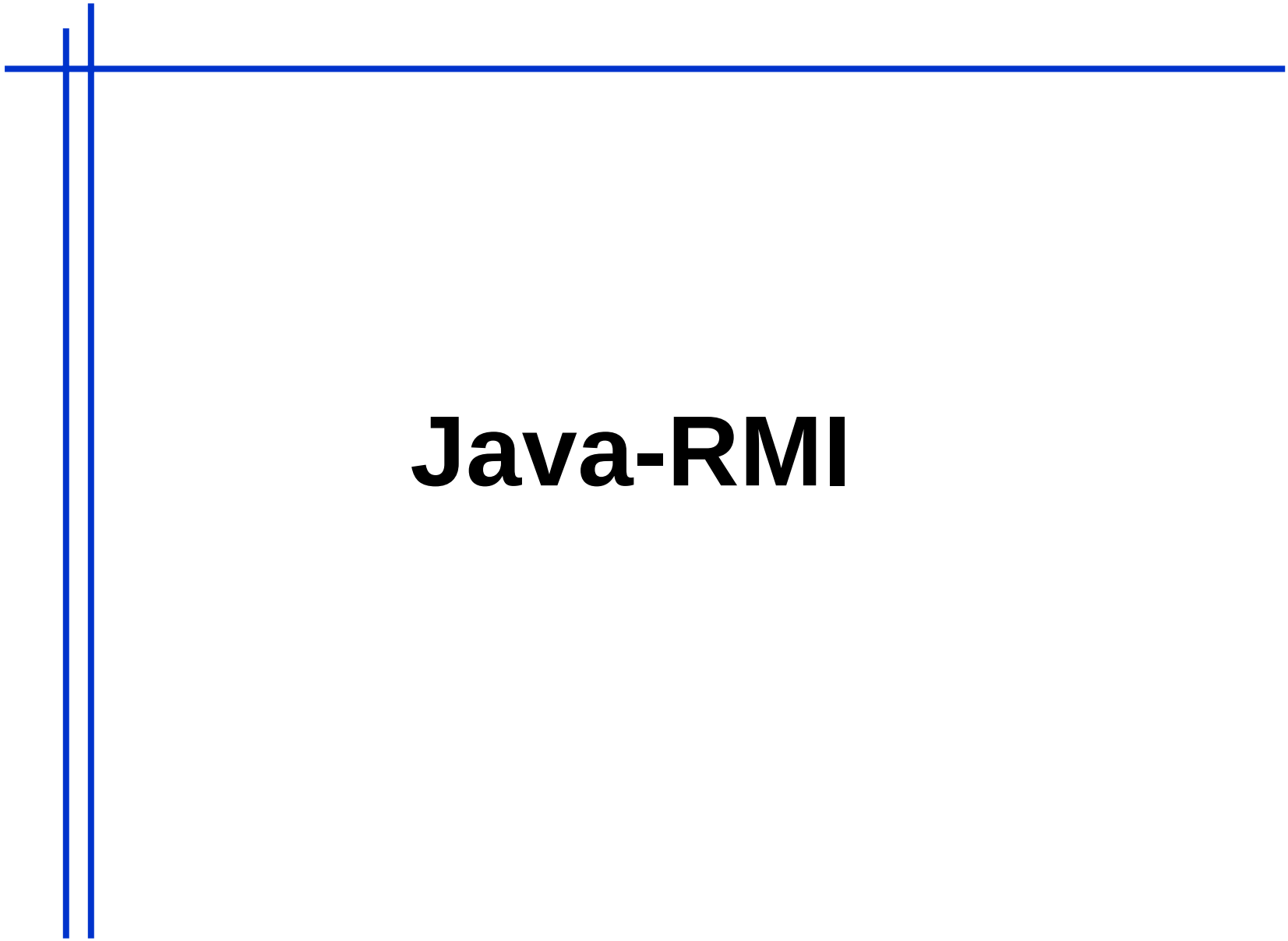
Implémentation

```
public class Presence extends Config implements Precursor
{
  final public IdentifierWrapper wrapper;
  public boolean evaluated = false;
  public Identifier event;
  public boolean posted = false;

  public Presence(IdentifierWrapper wrapper){ this.wrapper = wrapper; }
  public boolean fixed(){
     if(evaluated == false){ event = wrapper.evaluate(env); evaluated = true; }
     if(!(env.isGenerated(event) || env.eoi||posted)){
         env.getEventData(event).postPrecursor(this); posted = true;
     }
     return env.isGenerated(event) || env.eoi;
  }
  public boolean eval(){ return env.isGenerated(event); }
  public void zapFromHere(){
      if(parent instanceof Instruction){
          ((Instruction)parent).zap(null); posted = false;
      }
      else{ ((Config)parent).zap(); }
  }
  public void zap(){
      throw new InternalError("Illegal zap on Presence Config.");
  }
}
```

Implémentation

```
public class EventDataImpl implements EventData, Cloneable
{
   public long generated = 0, lastActualization = 0;
...
   public void generate(long instant,Object val){
      this.generated = instant;
...
      zapPrecursors();
   }
   public boolean isGenerated(long instant){
      return instant == generated;
   }
   PrecursorCell precursorList = null;

   public void postPrecursor(Precursor precursor){
       PrecursorCell cell = new PrecursorCell();
       cell.next = precursorList; precursorList = cell; cell.precursor = precursor;
   }
   public void zapPrecursors(){
       if(null == precursorList){ return; }
       while(null != precursorList){
           precursorList.precursor.zapFromHere();
           precursorList = precursorList.next;
       }
   }
}
```

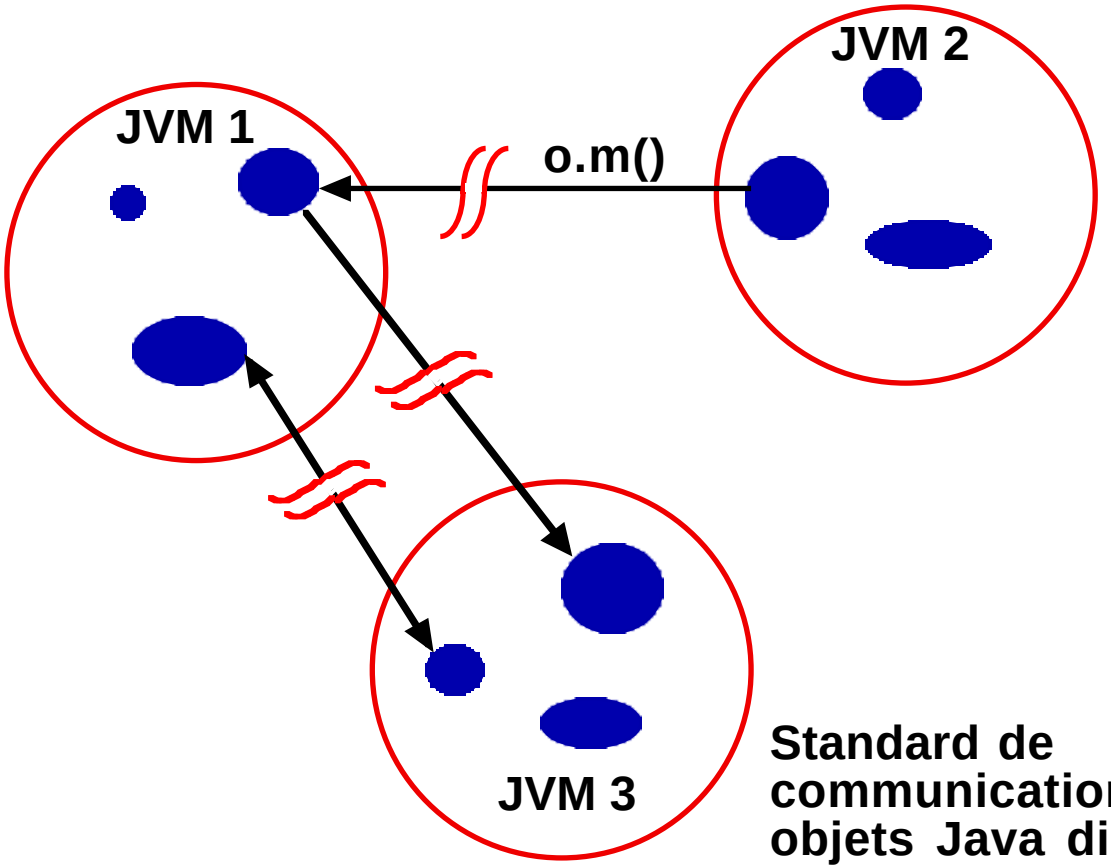
```
class PrecursorCell{
    PrecursorCell previous;
    PrecursorCell next;
    Precursor precursor;
}
```



Java-RMI

Java RMI

Remote Method Invocations

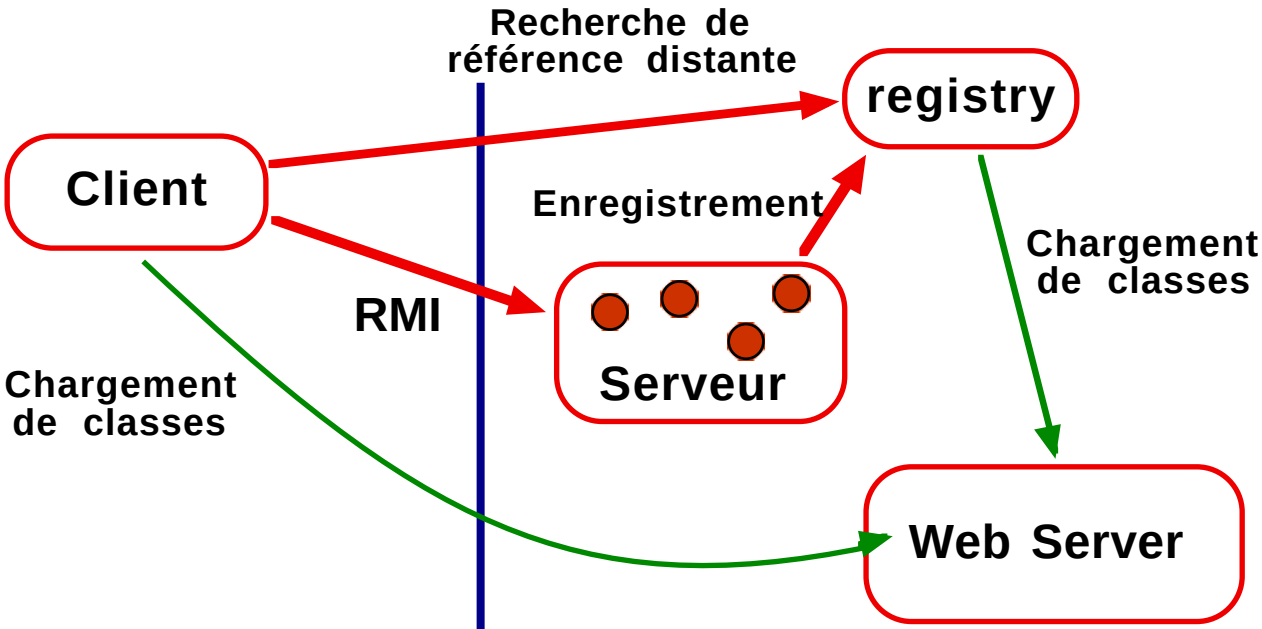


Standard de communication entre objets Java distants

Java RMI

- Architecture client-serveur
- Serveurs et clients sont des objets
- Communication par invocation de méthode synchrone type RPC
- Invocation transparente
- Permet l'échange d'objet ou de références distantes entre sites
- Utilise un serveur http pour gérer le téléchargement de code
- Permet l'implémentation de politiques de sécurité personnalisées

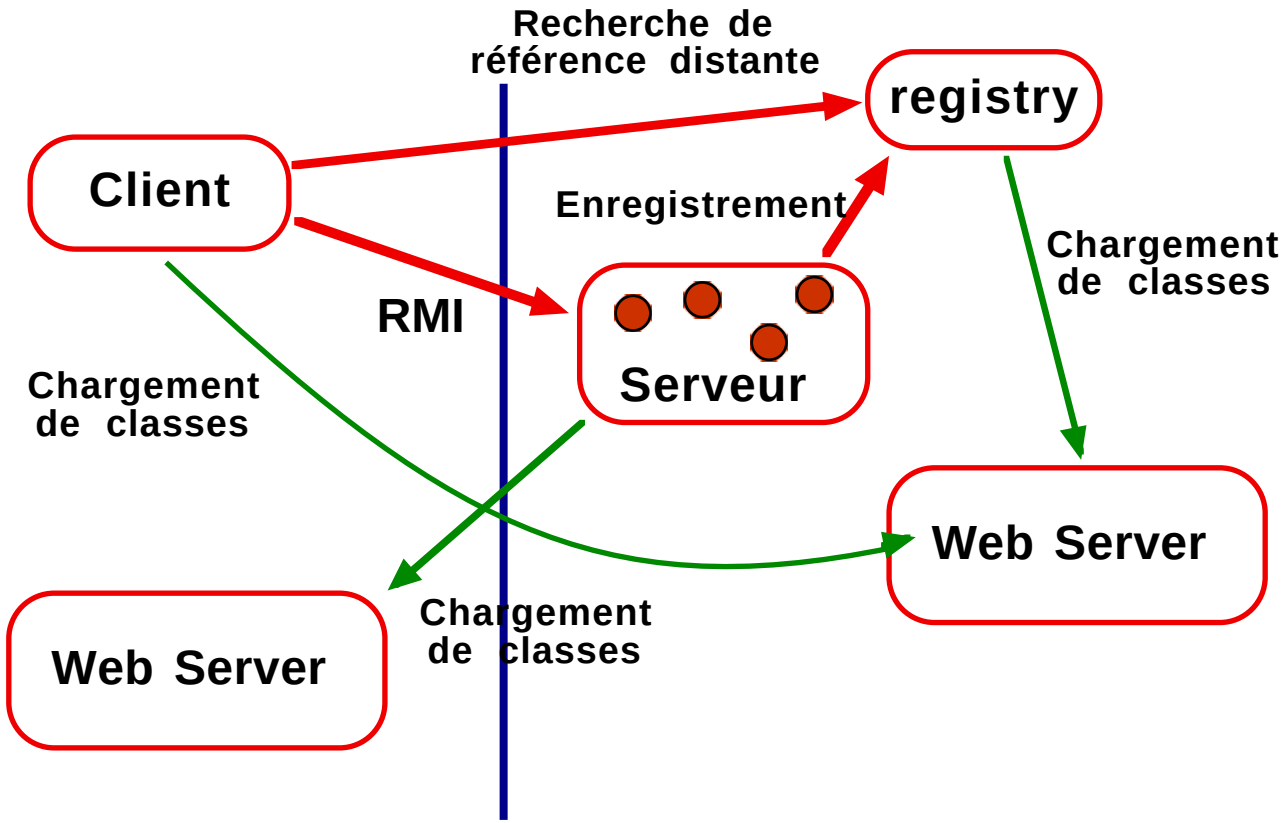
RMI : Application distribuée



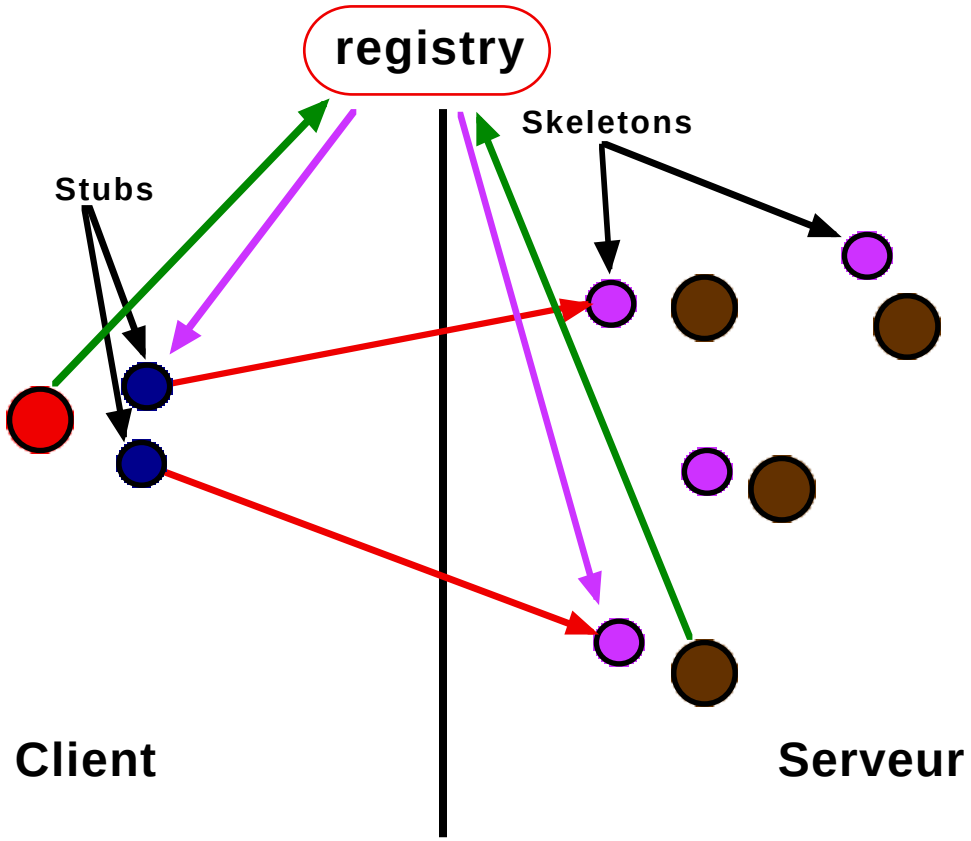
Services :

- **recherche de références distantes**
- **chargement de dynamique de bytecode**
- **communication par invocation de méthodes**

RMI : Application distribuée



Références distantes



Serveur

On définit une interface pour le serveur :

```
import java.rmi.*;
import inria.meije.rc.sugarcubes.*;

public interface RemoteMachine extends Remote
{
  public void addProgram(Program program)  throws RemoteException;
}
```

L'interface doit étendre Remote

Exception levée en cas de problème durant l'invocation de la

Implémentation du serveur :

```
public class RemoteMachineImpl extends
java.rmi.server.UnicastRemoteObject implements RemoteMachine
{
    protected Machine itsMachine = SC.machine();
...
    public void addProgram(Program program)  throws RemoteException{
        itsMachine.addProgram(program);
    }
}
```

Implémente les services de bases d'un serveur RMI

Un Programme est un objet sérialisable

Génération des Stubs et des Skeletons

Les références distantes sont implémentées par des objets, servant à masquer le caractère distant d'un objet.

Un couple d'objet Stub/Skeleton est utilisé. Leur génération est automatisée par rmic à partir de l'implémentation du serveur :

```
rmic RemoteMachineImpl
```

```
RemoteMachineImpl_Skel.class

RemoteMachineImpl_Stub.class
```

Serveur

Mise en route du serveur :

n° de port standard du registry RMI

Enregistrement du serveur dans le registry

Création du Skeleton correspondant sur le

```
try{
    if(1099 == portNumber){
        Naming.rebind("//"+hostName+"/"+name,server);
    }
    else{
        Naming.rebind("//"+hostName+portNumber+"/"+name,server);
    }
}
catch(Exception e){
    System.out.println("RsiServer error:"+e.getMessage());
    e.printStackTrace();
}
```

Client

On recherche une référence distante au serveur :

Création du Stub local

```
try{
    if(1099 == portNumber){
        target = (RemoteMachine)Naming.lookup("//"+hostName+"/"
                                              +targetName);
    }
    else{
        target = (RemoteMachine)Naming.lookup("//"+hostName
                                  +portNumber+"/"+targetName);
    }
}
catch(Exception e){
    System.err.println("Client Error:"+e.getMessage());
    e.printStackTrace();
}
```

Utilisation de la référence :

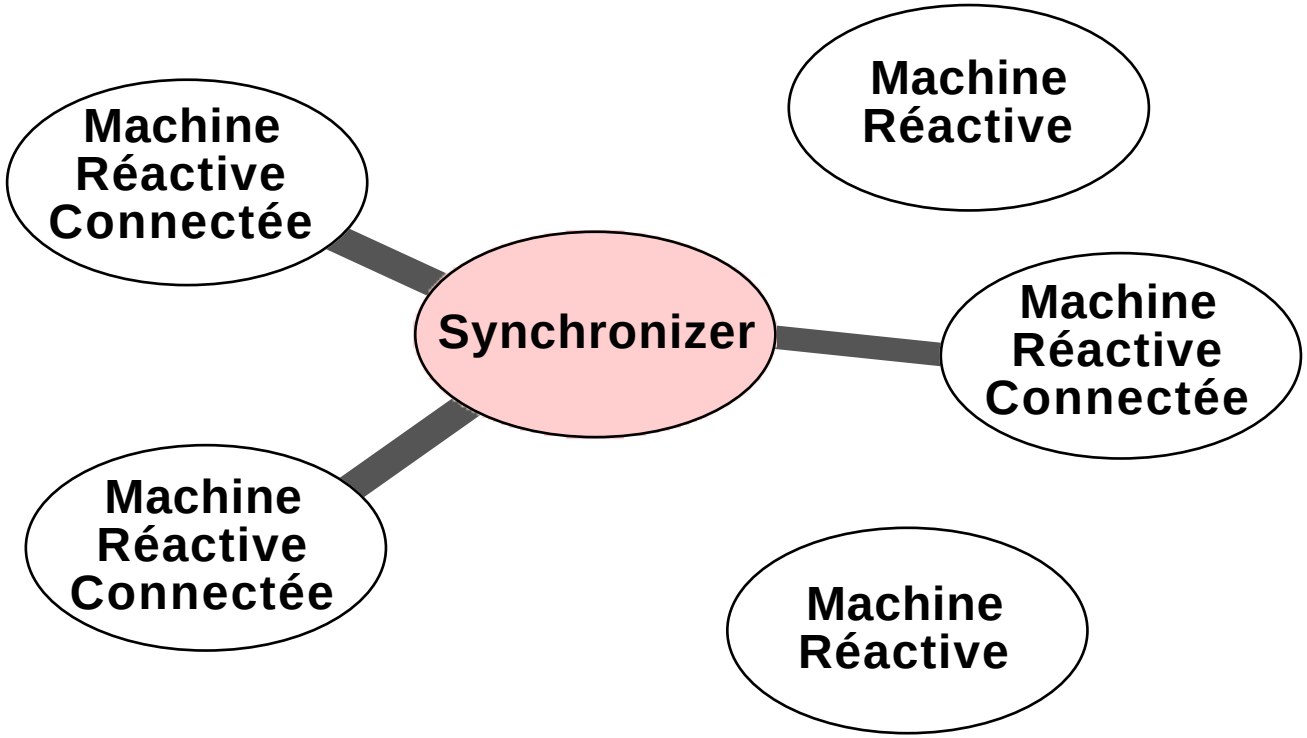
```
try{
   Program p = SC.seq(SC.await("e"),SC.print("hello"));
   target.addProgram(p);
} catch(Exception e){ e.printStackTrace(); }
```

Sérialisation

- Mécanisme transformant un objet Java en un flux d'octets
- Un objet est sérialisé avec l'ensemble des objets qu'il référence
- Le mot clé `transient` permet de limiter l'effet de la sérialisation
- Les types de primitifs du langage sont sérialisables ainsi que la plupart des objets de l'API (ex : composants AWT)

Les instructions Junior ou SugarCubes sont sérialisables

Machines Réactives Synchronisées

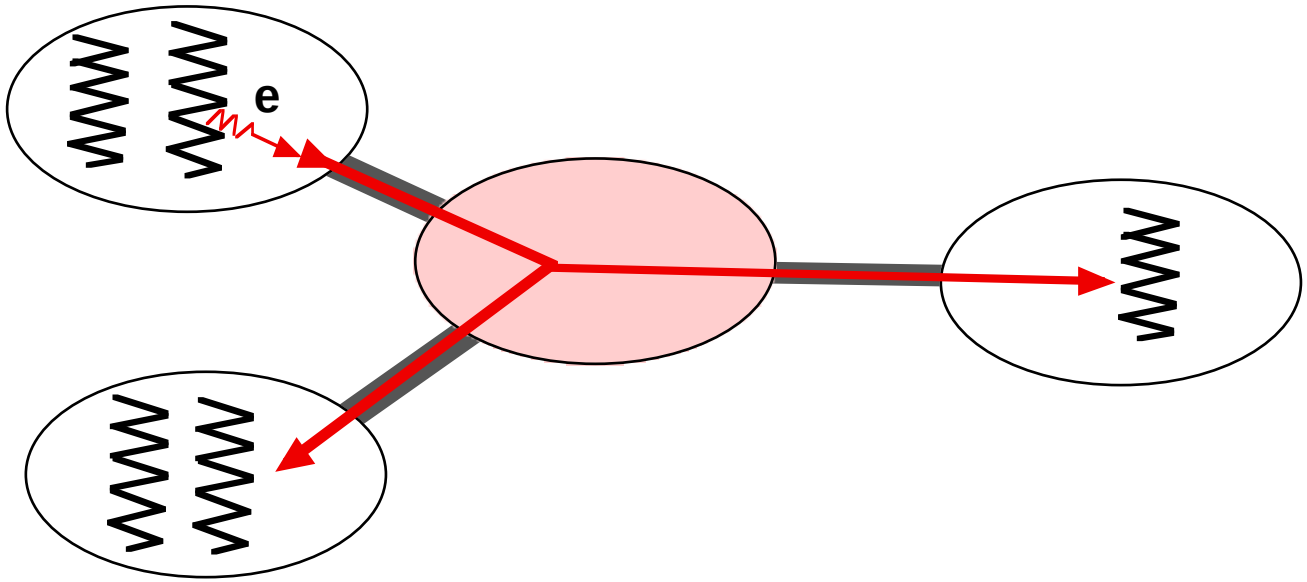


Zone d'exécution réactive distribuée

Notion d'intants partagée à travers le réseau

Connexion et déconnexion dynamique

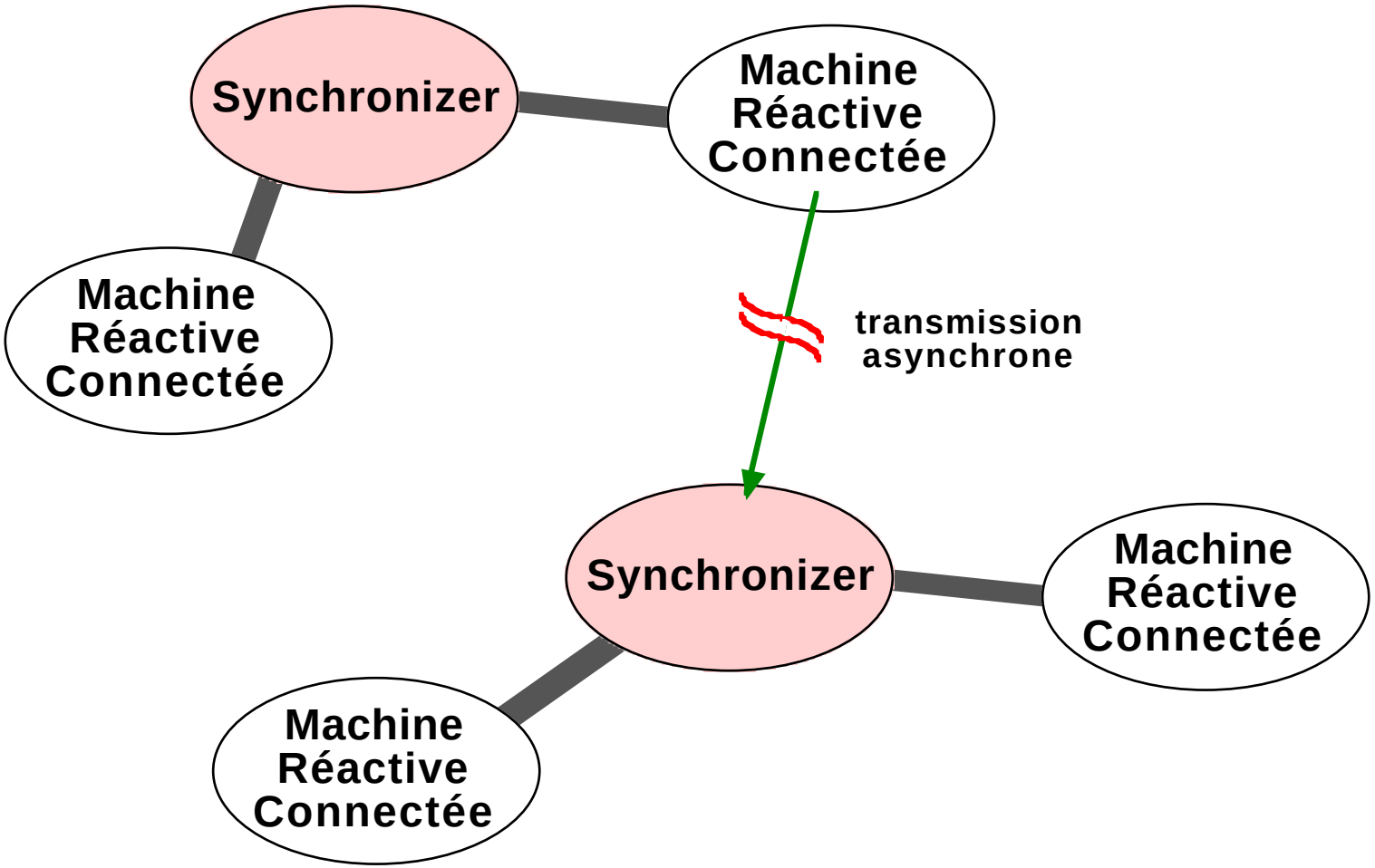
Diffusion d'événements à travers le réseau



Le synchroniseur diffuse les informations de génération

L'information sur l'absence de tous les événements non générés au cours d'un instant est factorisée avec la notification de fin d'instant.

Diffusion asynchrone



Implémentation

synchroniseur : algorithme de terminaison des instants distribuée.

Deux algos proposés :

- **Two phase Comit**
- **à base de compteurs**

Deux plateformes de distribution :

- **RMI**
- **Jonathan**

Synchronizer

```
public interface Synchronizer extends Remote
{
  public int connect(MachineSync mach)  throws RemoteException;
  public void disconnect(int id)        throws RemoteException;

  public void broadcast(String event)   throws RemoteException;

  public void suspended(int id)         throws RemoteException;
  public void completed(int id)         throws RemoteException;
}
```

connexion et déconnexion dynamique de machine

identificateur de la machine

génération d'événement à travers le réseau

la machine est suspendue

la machine a terminé pour l'instant courant

Synchroniseur

```
public interface Synchronizer_Impl extends Remote
{
  int numberOfMachines = 0;
  MachineSync machine[] = new MachineSync[MaxMachineNumber];
  byte status[] = new byte[MaxMachineNumber];

  Vector broadcastDemands = new Vector();
  Vector broadcastSum = new Vector();
}
```

nb de machines connectées

tableau des machines et leur statut

Vecteur des événements à diffuser au cours d'une phase

Toutes les demandes de diffusion depuis le début de

Diffusion des événements

```
public void broadcastProcessing(){
  for(int i = 0; i < MaxMachineNumber; i++){
    if(status[i]!=disconnected && status[i] != completed){
      status[i] = undef;
      try{ machines[i].generate(broadcastDemands);}
      catch(Exception e){...}
    }
  }
  broadcastDemands.removeAllElements();
}
```

Synchroniseur

Résolution d'un instant distribué

```
synchronized void step(){
  for(int i = 0; i < MaxMachineNumber;i++){
    if(status[i] == undef) return;
  }
  if(broadcastDemands.size() > 0){
    broadcastProcessing();
    return;
  }
  broadcastSum.removeAllElements();
  InstantIsOver();
}
```

Toutes les machines ont retourné leur statut

Il n'y a plus rien à diffuser

fin de l'instant

Machine d'exécution distribuée

```
public interface MachineSync extends Remote
{
  public void instantIsOver()             throws RemoteException;
  public void generate(Vector eventList) throws RemoteException;
}
```

Génération locale des événements diffusés

```
public void generate(Vector eventList){
  while(eventList.size() > 0){
    generate((String)eventList.firstElement());
    eventList.removeElementAt(0);
  }
  synchronized(this){ move = true; notifyAll(); }
}
```

Nouvelles instructions réactives

Trois nouvelles instructions réactives ont été introduites :

- **Connect** permet de demander une connexion à un synchroniseur et se comporte comme un Stop.

- **Disconnect** permet de demander une déconnexion et se comporte là encore comme un Stop

- **Broadcast** permet de générer des événements diffusés à travers un synchroniseur à l'ensemble des machines connectées

Conclusion

Un programme réactif peut-être physiquement distribué et conserver une notion d'instant global.

Les zones réactives distribuées peuvent être dynamiquement reconfigurées afin de ne pas trop pénaliser les modules les plus autonomes.

`http://www.inria.fr/mimosa/rp/`