# ALASCA — Architecture logicielles avancées pour les systèmes cyber-physiques autonomiques

**© Jacques Malenfant**

Master informatique, spécialité STL – UFR 919 Ingénierie

Sorbonne Université
Jacques.Malenfant@lip6.fr

# Cours 4

# Simulation des systèmes cyber-physiques

# Objectifs pédagogiques du cours 4

- Comprendre la **simulation** sur ordinateur, ses principes, ses applications.
- Comprendre comment la simulation permet de modéliser puis d'**exécuter virtuellement** un système pour mieux le comprendre, le mettre au point et l'optimiser.
- Distinguer et comprendre les trois grandes familles de modèles de simulation : la **simulation par événements discrets**, la **simulation continue** et la **simulation hybride**.
- Comprendre les principaux **algorithmes d'exécution** des simulateurs selon leur famille de modèle.
- Comprendre les relations entre les familles de modèles et leur capacité à **se modéliser les uns les autres** ainsi que leur capacité à **être intégrés dans la simulation hybride**.

# Plan

# Modèles de simulation

- Comme d'autres formes de modélisation, les *modèles de simulation* cherchent à représenter le comportement d'un système mais il s'agit dans ce cas de *modèles exécutables*.
- Intrinsèquement, simuler vise à *imiter* le comportement du système en l'*exécutant virtuellement* de manière à l'analyser à moindre coût qu'en expérimentant avec le système réel.
- Concrètement, simuler consiste à calculer l'état courant et les valeurs des variables sur une période de temps simulée donnée :
  - pour les transitions discrètes, il s'agit de calculer les valeurs des variables d'état modifiées par celles-ci ;
  - pour les évolutions continues, il s'agit de calculer les valeurs des variables en fonction du temps (par discrétisation, voir plus loin).
- Le niveau de détails du modèle (événements, variables) et la précision des calculs sont les garants du *degré de réalisme* d'un modèle de simulation et de la *fidélité des résultats à la réalité*, mais ils augmentent sa *complexité* et les *temps de calcul*, d'où la nécessité d'un *compromis*.

# Principaux types de modèles de simulation

- Confrontée à des équations différentielles sans solution analytique, calcul de la trajectoire continue (discrétisée) du modèle par l'intégration numérique de leurs équations : c'est la ***simulation continue***.
- En parallèle, confrontée aux systèmes discrets, calcul des modifications de l'état du modèle uniquement aux instants d'occurrences d'événements : c'est la ***simulation par événements discrets***.
  - La nature *stochastique* de plusieurs phénomènes du monde réel (arrivées de clients, durée de tâches, etc.) a été intégrée beaucoup plus tôt dans la simulation par événements discrets que dans la simulation continue.
- Confrontée à des systèmes combinants évolutions discrètes et continues, utilisation conjointe des deux approches précédentes pour donner la ***simulation hybride***.
  - Interprétation continue de l'évolution discrète : fonctions continues *en escalier*.
  - Principale difficulté : détection *précise* de l'instant d'occurrence des événements déclenchés par le *franchissement de frontières* par les variables continues (sauts et commutations de Branicki).

# Campagnes de simulation

- Calcul pas-à-pas d'***une*** *trajectoire* sur le temps simulé *i.e.*, un seul *exemple* du comportement défini par le modèle en fonction :
  - de ses données d'entrée et de son état initial,
  - et dans le cas stochastique, des réalisations particulières des phénomènes aléatoires qu'elle exhibe,

  qui n'est pas *individuellement* représentatif du comportement du système *en général*.
- Pour comprendre un système, il faudra donc étudier l'ensemble ou un *nombre représentatif* de ses trajectoires possibles.
- *Campagne de simulation* : *échantillonner* les trajectoires possibles sur un éventail *représentatif* de données d'entrée, d'états initiaux et de réalisations des phénomènes stochastiques qui permettra d'en analyser les résultats *spécifiques aux types de trajectoires* ou globalement *en moyenne*.
  - Plus un modèle de simulation est complexe, plus son exécution exige de calculs *i.e.*, *compromis* entre fidélité et temps de calcul !

# Quelques principes et concepts généraux

- En résumé, trois principales formes de simulation :
  1. Simulation par événements discrets
  2. Simulation continue
  3. Simulation hybride
- Mais aussi trois grandes modalités :
  1. simulation déterministe versus stochastique,
  2. simulation séquentielle/centralisée versus concurrente/répartie,
  3. simulation en temps simulé — logique, versus en temps réel – physique (ou en temps réel accéléré) $\Rightarrow$ *on y revient !*
- Principaux concepts de mise en œuvre :
  - horloge de simulation ;
  - modèle(s) et paramètres pour calculer les trajectoires aux instants nécessaires et suffisants dans le temps simulé (occurrences d'événements, instants de discrétisation $\Rightarrow$ *on y revient !*) ;
  - collecte des mesures utiles comme résultats de la simulation ;
  - la pluralité des états initiaux possibles et les phénomènes stochastiques requièrent d'échantillonner les trajectoires $\Rightarrow$ campagnes pour obtenir des analyses complètes et précises.

# Plan

# Qu'est-ce que la simulation par événements discrets ?

- Les systèmes à événements discrets sont des systèmes dont l'état n'*évolue significativement* qu'aux *instants d'occurrence* d'événements se produisant en *temps continu*.
  Par exemple, en simulant une file d'attente, la longueur de la file ne change qu'aux instants où des clients arrivent et où ils débutent leur service ; de même, leur durée totale de passage n'est calculée qu'à l'instant où leur service se termine.
- Puisque leur état ne change qu'aux occurrences d'événements, on peut se contenter de *calculer les valeurs des variables du modèle qu'aux instants d'occurrence des événements*.
  Par exemple, sur la file d'attente, il suffit de calculer la longueur de la file qu'aux instants d'arrivées et de débuts de service, puis la durée de passage qu'aux instant de fin de service pour connaître toute la trajectoire.
- Les algorithmes de base de simulation par événements discrets utilisent cette propriété pour calculer la trajectoire en *passant d'un événement au suivant et avançant l'horloge par sauts, d'un instant d'occurrence au suivant*.

# Modalités de simulation par événements discrets

- Il existe deux grandes familles d'algorithmes d'exécution des simulations par événements discrets :
  1. *Basique* : à chaque occurrence d'événements le modèle détermine (uniquement) le *prochain* événement et son instant d'occurrence ; le moteur de simulation avance l'horloge à cet instant pour l'exécuter puis recommence.
  2. *Avec ordonnancement d'événements* : le modèle peut engendrer plusieurs événements futurs ordonnancés dans une liste ordonnée par instants d'occurrence croissants ; le moteur de simulation avance d'un événement au suivant en prenant à chaque fois le prochain (premier) événement dans la liste.
- La modalité *basique* est très simple pourtant elle permet d'exprimer toutes les formes de simulation par événements discrets (ainsi que la discrétisation de la simulation continue).
- La seconde modalité a été énormément utilisée (voir le livre de Law dont la référence est donnée au dernier transparent).

# Modèles de simulation avec ordonnancement d'événements

Parmi les modèles avec ordonnancement, il y a aussi deux approches duales :

- Des modèles *axés événements exécutables* sur entités passives :

  idée : modéliser le comportement du système par des *événements* dont l'exécution modifie l'état du système ;
  simuler : exécuter la séquence des événements les uns après les autres pour calculer la trajectoire.

  *Exemple : clients et guichets d'une banque sont simulés en exécutant les événements « arrivée », « début » et « fin de service ».*

- Des modèles *axés entités exécutables* à événements implicites :

  idée : modéliser le omportement du système par celui des *entités*, exprimé par des actions exécutées à des instants ponctuels ;
  simuler : exécuter en parallèle toutes les entités du modèle et leurs actions pour calculer la trajectoire.

  *Exemple : l'entité « client » arrive à la banque et s'insère dans la file, attend, se présente au guichet, est servi, termine et part.*

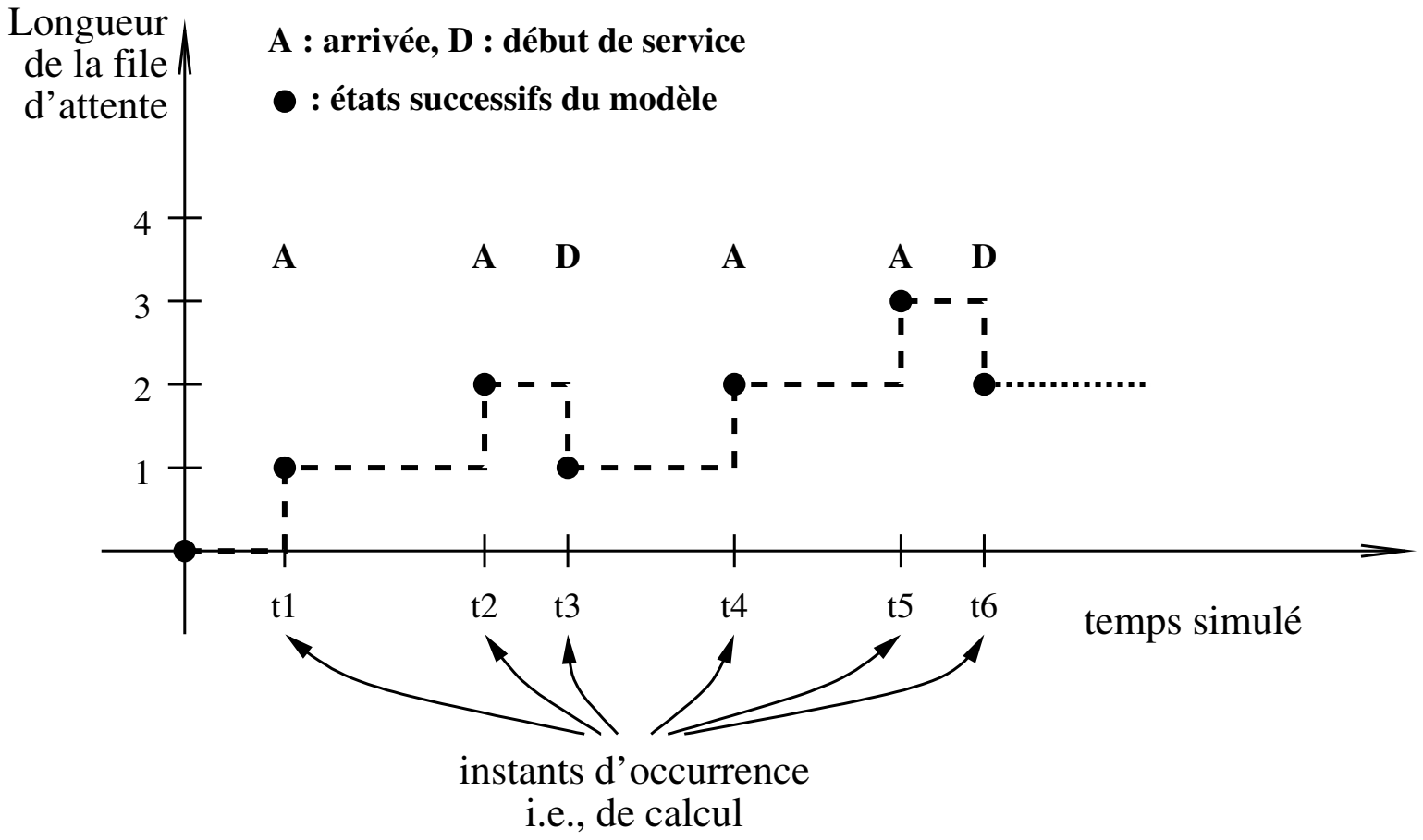
*Nota : très proche de l'idée fondamentale de la programmation par objets, le premier langage, Simula, ayant servi à faire de la simulation par événements.*

# Modèles axés événements exécutables explicites

- Composés de types d'événements et d'*entités passives* qui :
  - ont des propriétés capturées par les valeurs de leurs *attributs* et
  - vont être affectés par des occurrences d'événements changeant leur état.

  La collection de tous les attributs de toutes les entités ainsi que les variables globales du modèle forment l'*espace d'états*.
- Les *occurrences* d'événements sont les moments où un prochain état est calculé et de nouveaux événements engendrés.
  - Les événements débutent ou terminent des actions, observées sur le système réel, d'une certaine *durée* laquelle n'est pas matérialisée dans la simulation car *rien de significatif* nécessite d'être observé sur l'état pendant celle-ci.
- Le comportement d'un système est donc simulé en calculant la trajectoire des valeurs d'attributs des entités et des variables globales au fil d'un temps simulé discret car *restreint* aux instants d'occurrences des événements.
  - Dans un modèle *déterministe*, tous les événements, leurs relations causales et leurs impacts sur l'état sont déterministes, alors que dans un modèle *stochastique* certaines de ces relations ou de ces impacts sont aléatoires.

# Exemple : calcul de la longueur de la file d'attente



# Étapes de modélisation axée événements I

1. Identification des *types d'entités* avec l'élicitation de leurs *propriétés* et définition de leurs *attributs*.
   *Ex.: file* $\rightarrow$ *longueur.*
2. Identification *types d'événements* ponctuant le déroulement de l'exécution du modèle *i.e.*, des activités des entités impliquées.
   *Ex.: arrivée d'un client, début de service d'un client, . . .*
3. Description de la façon dont l'exécution des types d'événements change l'état des entités ou du modèle.
   *Ex.: début de service d'un client* $\Rightarrow$ *retire le premier élément de la file, passe le guichet à l'état occupé, . . .*
4. Identification et description *relations causales* entre les événements pour engendrer les nouveaux événements et leurs instants d'occurrence (avec les lois de probabilités et leurs paramètres).
   *Ex.: le début de service d'un client causant sa future fin de service celle-ci causant le début de service du client suivant.*

# Étapes de modélisation axée événements II

- Ajout éventuel de relations causales *synthétiques* pour faciliter la simulation.
  *Ex.: arrivée d'un client qui engendre l'arrivée suivante du prochain client.*

5. Programmation : chaque évenement peut « s'exécuter » en modifiant l'état et en engendrant de nouveaux événements au besoin.
   *Ex.: algorithme d'exécution du début de service d'un client*

**Data :** $q$, file d'attente des clients
**Data :** $c$, client
**Data :** $g$, guichet
1 $c \leftarrow q.retire()$;
2 $g.occupe \leftarrow vrai$;
3 $d \leftarrow$ générer une durée de service pour $c$;
4 créer et ordonnancer l'événement fin de service pour $c$ dans $d$ unités de temps;

# Moteur de simulation axé événements basique

- Modèle proposant deux fonctions donnant :
  1. le délai d'attente jusqu'à l'occurrence du prochain événement ;
  2. l'état résultant de l'exécution des événements sur l'état courant.
- Algorithme d'exécution : soit $t$ le temps simulé selon l'horloge,
  1. Récupérer le délai $d$ jusqu'à l'occurrence du prochain événement.
  2. Avancer l'horloge de simulation à $t + d$.
  3. Exécuter le prochain événement pour obtenir un nouvel état.
  4. Tant que la simulation n'est pas terminée, reprendre à 1.
- Gestion du temps simulé et de l'horloge de simulation :
  - en temps logique = extraction des occurrences d'événements du temps simulé continu pour retenir une séquence d'instants de calcul, exécutée par saut immédiat d'un instant au suivant ;
  - en temps réel : alignement de l'horloge simulée sur le temps réel, synchronisant instants de calcul avec instants du temps réel (avec mise en sommeil d'un instant au suivant) ;
  - en temps réel accéléré : contraction ou expansion du temps selon une relation linéaire constante entre temps simulé et temps réel.

# Moteur de simulation avec ordonnancement

- Si les modèles basiques ne voient que l'événement suivant à exécuter, les modèles à ordonnancement d'événements, eux, permettent de *planifier à l'avance* plusieurs événements à venir.
- File des événements à venir : ensemble des événements triés en ordre croissant de leurs instants d'occurrence $t_1, t_2, \ldots$.
- Algorithme d'exécution :
  1. Retirer le premier événement dans la file.
  2. Avancer l'horloge à l'instant d'occurrence de cet événement.
  3. Exécuter cet événement pour obtenir un nouvel état.
  4. Générer les événements futurs causés par cet événement et les insérer dans la file des événements.
  5. Tant que la file des événements n'est pas vide, reprendre à 1.
- Le moteur peut utiliser les mêmes modalités de gestion du temps simulé que les modèles de base.

# Production de trajectoires dans le cas stochastique

- Dans les simulations par événements discrets, l'aléa apparaît souvent dans les instants d'occurrence des événements.
  *Ex.: arrivées des clients au guichet.*
- On modélise souvent cette forme d'aléa comme une durée aléatoire entre les occurrences d'événements.
  *Ex.: delais entre les arrivées des clients.*
- Comment obtenir des valeurs concrètes pour ces délais ?
  1. On utilise des générateurs de nombres pseudo-aléatoires[1] qui, à partir d'une valeur initiale choisie (germe), engendrent une séquence de nombres suivant une certaine densité de probabilité.
  2. Pour chaque événement engendré, l'instant de son occurrence est obtenu en utilisant le prochain nombre dans la séquence obtenue du générateur.

  *Ex.: au début de service d'un client, on engendre sa fin de service après un délai aléatoire qui suit une loi exponentielle.*

[1] Presque tous les langages de programmation offrent de tels générateurs, mais ils sont rarement bons ; il vaut mieux utiliser des bibliothèques comme common-maths.

# Exemple de système discret : exécution de requêtes I

- Modèle de file d'attente M/M/1 : arrivées markoviennes (*i.e.*, aléatoires), temps de service markoviens, un serveur.
- Entités : requête, machine virtuelle, file d'attente
- Attributs :
  - requête : moment d'arrivée
  - machine virtuelle (MV) : libre ou occupée (en train d'exécuter une requête)
  - file : séquence des requêtes en ordre d'arrivée
- Événements et leur exécution :
  - arrivée d'une requête : marque la requête par le temps simulé de son arrivée, l'ajoute à la fin de la file d'attente et si la MV est libre, provoque immédiatement un début de service pour cette requête.
  - début de service : retire la première requête de la file, passe la MV à occupée et génère l'événement de fin de service pour la requête retirée.

# Exemple de système discret : exécution de requêtes II

- fin de service : passe la MV à libre, calcule le temps de service de la requête puis, si la file n'est pas vide, provoque immédiatement le début de service de la prochaine requête.

- Causalité synthétique entre événements :
  - Les arrivées peuvent engendrer des débuts de service, lesquels engendrent les fins de service, qui elles-mêmes peuvent engendrer un début de service, mais comment engendrer les arrivées ?
    - Solution 1 : générer à l'avance toutes les arrivées et les insérer dans la file des événements.
      $\Rightarrow$ Long et coûteux en espace pour les simulations longues.
    - Solution 2 : générer la première arrivée, et ajouter une relation causale synthétique faisant que le traitement d'un évenement arrivée génère l'événement arrivée suivant.
      $\Rightarrow$ Beaucoup moins coûteux en espace.

# Animation du modèle

**Horloge : 0**

**File des événements**

| | |
|---|---|
| Arrivée Requete1<br>t : 2 | |

Requete1
arrivée :

**File d'attente**

Machine virtuelle
état : libre

Somme des temps :
nombre de requetes :

# Animation du modèle

**Horloge : 2**

**File des événements**

| | | |
|---|---|---|
| Début de service Requete1<br>t : 2 | Arrivée Requete2<br>t : 3 | |

| | |
|---|---|
| Requete1<br>arrivée : 2 | Requete2<br>arrivée : |

**File d'attente**

| | |
|---|---|
| | |

| |
|---|
| Machine virtuelle<br>état : libre |

Somme des temps :

nombre de requetes :

# Animation du modèle

**Horloge : 2**

**File des événements**

| | | |
|---|---|---|
| Arrivée Requete2<br>t : 3 | Fin de service Requete1<br>t : 5 | |

| | |
|---|---|
| Requete1<br>arrivée : 2 | Requete2<br>arrivée : |

**File d'attente**

| |
|---|
| |

| |
|---|
| Machine virtuelle<br>état : occupée |

Somme des temps :
nombre de requetes :

# Animation du modèle

**Horloge : 3**

**File des événements**

| | | |
|---|---|---|
| Fin de service Requete1<br>t : 5 | Arrivée Requete3<br>t : 6 | |

| | | |
|---|---|---|
| Requete1<br>arrivée : 2 | Requete2<br>arrivée : 3 | Requete3<br>arrivée : |

**File d'attente**

| | |
|---|---|
| | |

Machine virtuelle
état : occupée

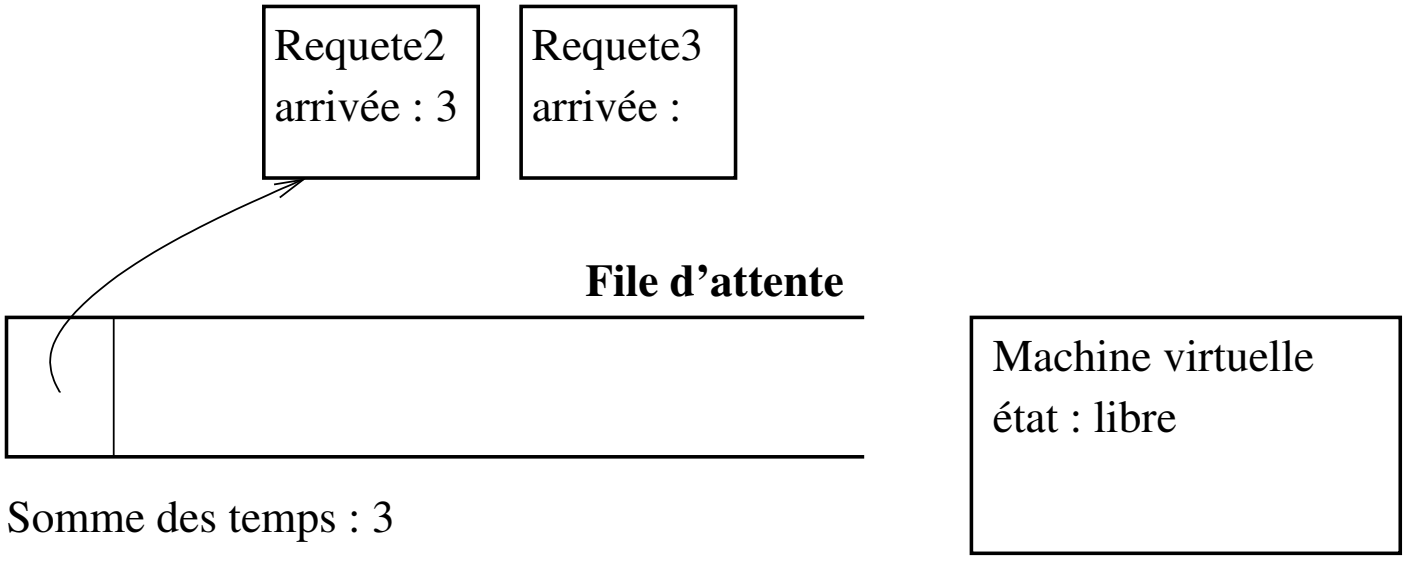
Somme des temps :
nombre de requetes :

# Animation du modèle

**Horloge : 5**

**File des événements**

| | | |
|---|---|---|
| Début de service Requete2<br>t : 5 | Arrivée Requete3<br>t : 6 | |




Somme des temps : 3
nombre de requetes : 1

# Animation du modèle

**Horloge : 5**

**File des événements**

| | | |
|---|---|---|
| Arrivée Requete3<br>t : 6 | Fin de service Requete2<br>t : 9 | |

| | |
|---|---|
| Requete2<br>arrivée : 3 | Requete3<br>arrivée : |

**File d'attente**

| |
|---|
| |

| |
|---|
| Machine virtuelle<br>état : occupée |

Somme des temps : 3
nombre de requetes : 1

- Et ainsi de suite jusqu'au temps de fin de simulation ou une liste d'événements vide...

# Événements déclenchés par changement d'états du modèle

- Dans la simulation par événements classique, les événements sont ordonnancés puis déclenchés par le temps *i.e.*, leur *moment d'occurrence* dans le temps simulé.
- Une extension autorise des événements déclenchés par des *modifications de valeur des variables* et ce selon deux modalités :
  - dès qu'une modification est *observée*, peu importe laquelle ;
  - dès que la ou les valeurs d'une ou plusieurs variables satisfont à une certaine *condition logique* (franchissement de frontière).
- Pour la simulation par événements discrets, puisque les valeurs des variables ne changent qu'aux instants d'occurrence des événements, il suffit :
  - de vérifier ces conditions lors de l'exécution des événements et
  - de générer les événements détectés pour *occurrence immédiate*.
- Difficulté : possibilité de chaînes de modifications entraînant plusieurs événements avec occurrence immédiate au même instant, ce qui doit doit bien sûr *toujours s'arrêter*.

# Effet Zeno

- « L'effet Zeno » se produit quand un système cherche à exécuter un nombre *potentiellement illimité* d'événements à un *même instant ponctuel*, le moteur de simulation ne pouvant alors plus faire progresser l'horloge.
  - L'effet Zeno peut potentiellement se produire dès lors qu'on autorise un modèle à proposer un événement qui peut causer d'autres événements *au même instant*, qu'ils soient engendrés avec un délai d'occurrence égal à 0 ou par une condition logique.
  - Mais c'est à la fois nécessaire, car certains systèmes exhibent de tels événements, et aussi très pratique pour simplifier la spécification de certains événements et modèles.
    Ex.: fin de service qui déclenche le début de service suivant.
- Ainsi, un modèle de simulation *correct* doit toujours s'assurer de ne pas pouvoir engendrer un nombre potentiellement illimité d'événements au même instant.
  - Un peu un équivalent en simulation de la boucle ou de la récursivité qui ne s'arrêtent pas en programmation.

# Exemples venant de notre fil rouge MolèNE

- Modèle basique :
  - Modèle Tic : chaque événement tic engendre uniquement le suivant après une durée fixe, paramètre du modèle.
- Modèle à ordonnancement d'événements :
  - Modèle d'interruptions du réseau : deux événements, interruption et reprise, s'ordonnancent l'un l'autre avec des délais aléatoires (durées) entre eux.

Pour les automates hybrides, la simulation par événements permet d'exécuter des comportements continus sans avoir à les simuler en temps continu :

- Modèle continu « réveil-matin » : le délai jusqu'au tic est modélisé par une valeur initiale $d = d_0$, une équation différentielle $d'(t) = -1$ et une condition frontière de transition $d = 0$.
- Beaucoup plus coûteux de simuler l'écoulement continu du temps pour détecter quand sonner que le faire par un événement...

# Plan

# Comment se fait la simulation continue ?

- (Rappel) Moteur de simulation continue : calcul numérique, et en particulier l'intégration numérique.
  - L'*intégration numérique* est une technique d'analyse numérique pouvant traiter n'importe quelle forme d'intégrale définie comme, par exemple, $y = \int_{x=x_0}^{x=x_1} f(x)dx$
  - Bien qu'à finalité continue, elle part toujours d'une *discrétisation*, compte tenu de la nature discrète de tout calcul sur ordinateur.
  - L'approche la plus courante consiste à *discrétiser l'abcisse* (ici $x$) et alors le procédé de base consiste à :
    - implanter la fonction $f(x)$ en code exécutable,
    - adopter un pas d'intégration $\Delta x$,
    - calculer l'aire sous la courbe de proche en proche en sommant $\Delta x$ par $\Delta x$ *e.g.*, $\sum_i f(x_i)\Delta x$, $x_i = x_0, x_0 + \Delta x, x_0 + 2\Delta x$, ...
- Vision simulation : intégration par rapport *au temps i.e.*, l'abcisse correspond à l'horloge qui progresse du temps initial $t_0$ au temps final $t_f$ par pas de $\Delta t$, en calculant à chaque fois une nouvelle valeur dans la trajectoire à $t_i, i = 0, \ldots, f$.

# Intégration numérique d'une équation différentielle

- Variante de l'intégration numérique où on part d'une équation différentielle $y'(t)$ qu'on implante en code exécutable pour calculer numériquement la dérivée de la fonction recherchée $y(t)$.
  - À partir d'une condition initiale $y(t_0) = y_0$, on calcule les valeurs successives de $y_{i+1}, i = 0, 1, \ldots$ à partir de la valeur précédente $y_i$ et d'un certain nombre de valeurs de $y'(t)$ calculées à chaque étape pour différents instants dans l'intervalle $[t_i, t_{i+1}]$.
  - Le délai entre les valeurs successives $\Delta t = t_i - t_{i-1}$ est appelé un *quantum de temps* servant de *pas d'intégration*.
- Selon la méthode d'intégration choisie, on utilise :
  - la valeur de la dérivée à $t_i$ *i.e.* $y'(t_i)$ ;
  - des valeurs intermédiaires dans l'intervalle selon la méthode d'intégration choisie (ex.: $t_i + \Delta t/4, t_i + \Delta t/2, t_i + 3\Delta t/4$).

Ici, on présente le pas d'intégration $\Delta t$ comme constant, mais plusieurs méthodes d'intégration numérique le font varier selon l'ampleur de la dérivée pour diminuer les erreurs de calcul (*méthodes adaptatives*).

# Intégration numérique par quantum de temps simulé

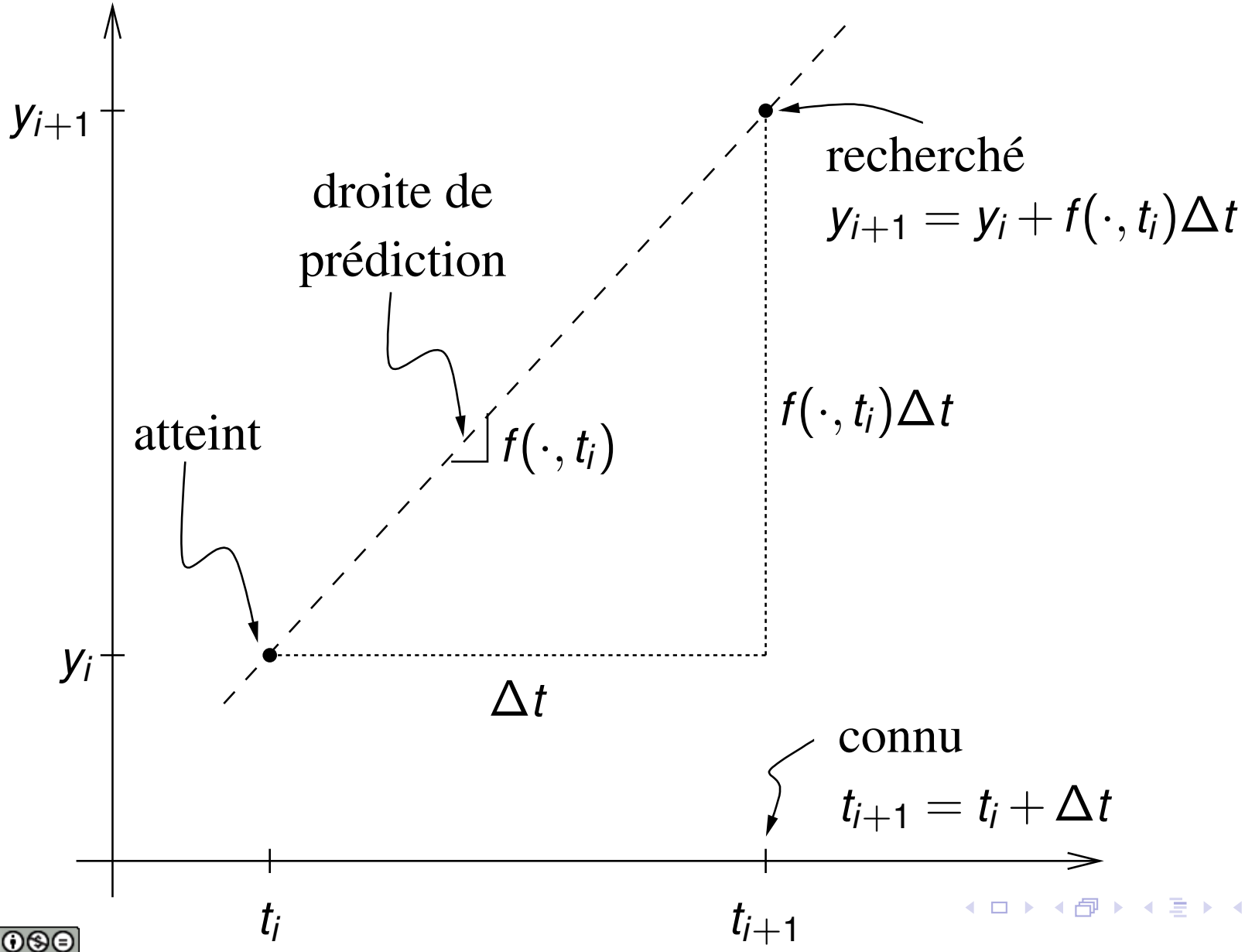
- La méthode d'Euler est la méthode d'intégration numérique par quantum de temps la plus ancienne et la plus simple.
- Elle utilise une approximation linéaire pour faire le calcul suivant :

$$y_{i+1} = y_i + y'_i \Delta t$$

où $\Delta t = t_i - t_{i-1}$ est le pas d'intégration choisi.

- Les avantages de la méthode d'Euler sont sa simplicité et la faible quantité de calcul qu'elle nécessite à chaque pas d'intégration.
- L'approximation linéaire d'Euler entraîne cependant une erreur de calcul assez importante en général.
  - On peut réduire l'erreur en diminuant le pas d'intégration.
  - Mais cette réduction augmente la quantité totale de calculs à faire et trop le réduire peut finir aussi par réaugmenter l'erreur.
- Lorsque la réduction du pas d'intégration devient trop importante pour obtenir une certaine précision, on peut passer plutôt à une méthode d'intégration non linéaire.

# Quantum de temps Euler (linéaire)



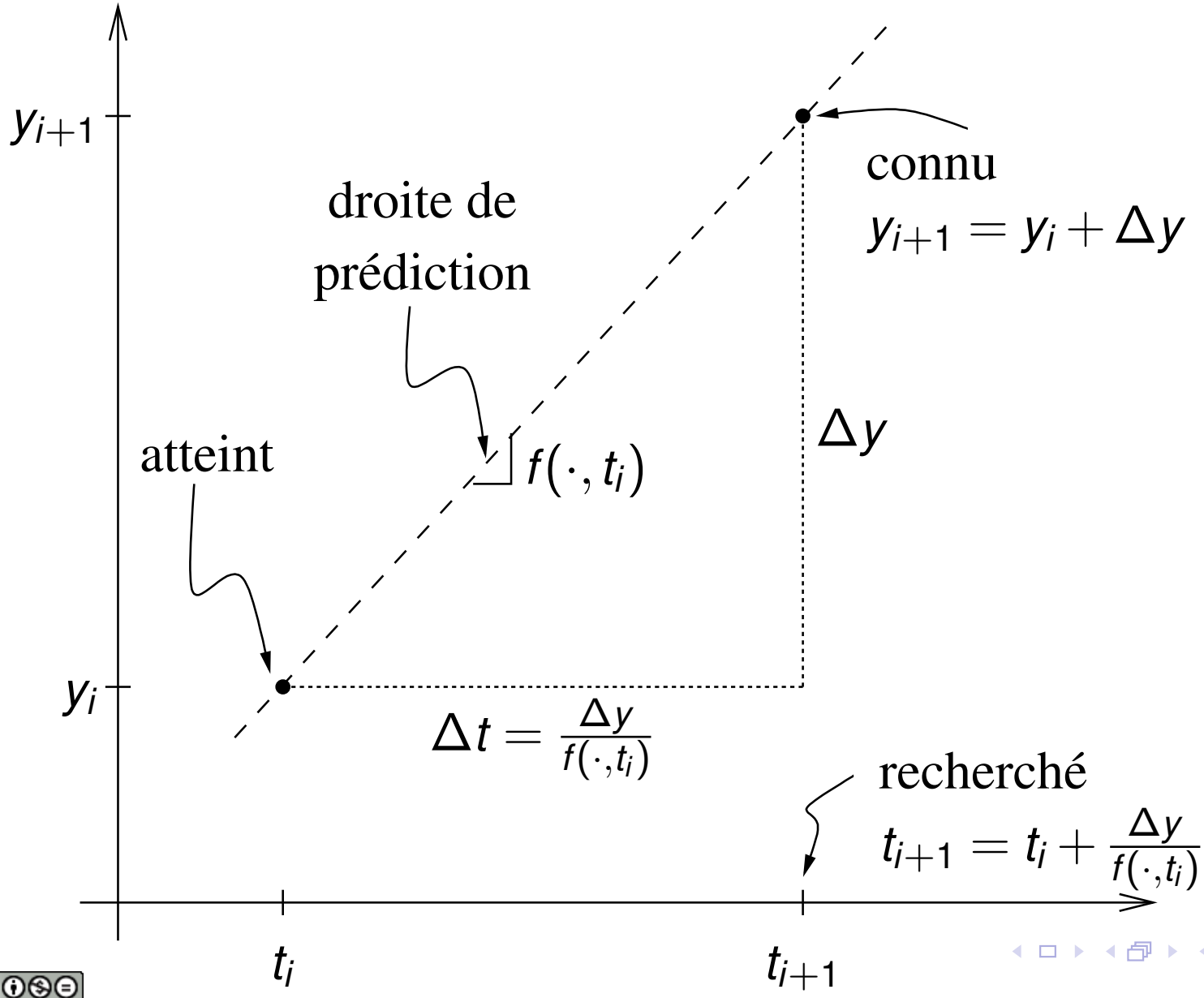
# Méthodes plus précises, mais plus coûteuses

- La *méthode d'intégration* est choisie en fonction du type d'équations différentielles et de la précision souhaitée.
  - Elles approximent plus ou moins bien la fonction calculée.
    - L'erreur ne dépend pas que de la méthode, mais aussi de la forme de la fonction (courbure) dont on doit tenir compte pour les choix de méthode et de pas d'intégration.
  - Choix $\Rightarrow$ *compromis* entre *précision* et *quantité de calcul*.
- Pour obtenir des résultats plus précis qu'Euler, d'autres méthodes utilisent une *approximation quadratique*.
- La méthode quadratique la plus populaire est aujourd'hui celle de *Runge-Kutta d'ordre 4*, appelée ainsi car elle utilise quatre valeurs de la dérivée à chaque pas d'intégration
  (multipliant d'autant la quantité de calcul par rapport à Euler où une suffit).
- D'autres méthodes font (aussi) varier le pas d'intégration en fonction de l'ampleur de la dérivée à l'étape courante.
- Des bibliothèques proposent des méthodes de résolution des équations différentielles (ex.: `org.apache.commons.math3.ode`).

# Approche originale : *quantized state systems*

- Rappel : discrétisation par *quantum de temps* :
  - On fixe un quantum de temps $\Delta t$ et à chaque itération on l'utilise comme pas d'intégration en se demandant :
    *Quelle sera la valeur de* $y_{i+1}$ au temps $t_{i+1}$ étant donnée sa valeur $y_i$ et sa dérivée $y'_i$ à $t_i$, où $t_{i+1} = t_i + \Delta t$.
- Dans plusieurs cas, on peut obtenir un meilleur contrôle de l'erreur pour un coût en calcul inférieur en utilisant un *quantum d'état*, technique appelée *quantized state systems* ou QSS.
  - On fixe un quantum de variation de l'état $\pm\Delta y$ et on se demande *à quel moment* $t_{i+1} = t_i + \delta_i$ la valeur $y_{i+1}$ va atteindre $y_i \pm \Delta y$ (selon le signe de la dérivée), $\delta_i$ devenant le pas d'intégration à cette étape de calcul.
  - C'est donc une technique à pas d'intégration *variable*.
- Pour prédire $\delta_i$, on utilise une approximation de la fonction $y$, comme dans les méthodes à quantum de temps. On trouve :
  - des prédictions linéaires, méthodes *QSS1* ;
  - des prédictions non linéaires, méthodes *QSS2*, *QSS3*, ...

# Quantum d'état QSS1 (linéaire)



# Quand choisir la discrétisation de l'état ?

- Pour réduire la quantité de calculs nécessaires :
  - Observation : plus la dérivée est faible, plus il faut de temps pour atteindre une variation de $\Delta y$ *fixe* sur la fonction $y$.
  - Ainsi, en QSS, plus les dérivées sont faibles, plus les pas d'intégration s'allongent et moins il y a de calculs à faire pour une durée totale de simulation fixée.
  - On choisit donc les approches QSS lorsque les valeurs de $y'(t)$ demeurent relativement faibles car, alors, elles permettent d'allonger les $\Delta_i$ et de diminuer la quantité de calcul pour la simulation.
  - Une approche *mixte* passe de QSS au quantum de temps en fonction de la valeur instantanée de la dérivée.
- Pour réduire et mieux contrôler les erreurs sur $y$ :
  - Fixer $\Delta y$ fait basculer l'erreur du côté de $\delta$ et donc sur le temps, ce qui est souvent plus facile à contrôler.
  - Les méthodes QSS permettent dans beaucoup de cas d'obtenir une meilleure précision que de fixer des quantums de temps, *à quantité de calculs constant*.

# Cas continu stochastique

Deux principaux types de phénomènes stochastiques continus :

- Variables aléatoires continues mais à *valeurs ponctuelles* réalisées par leurs fonctions de distribution de probabilité et dont les valeurs individuelles sont produites par des générateurs de nombres pseudo-aléatoires comme on l'a vu.
  - *Ex.: durée d'exécution d'une requête.*
- Trajectoires aléatoires (dont les mouvements browniens) modélisées par des équations différentielles stochastiques :
  - *Ex.: vol d'une abeille au-dessus d'un champ de fleurs*.
  - Elles peuvent être étudiées de deux façons :
    1. Production de trajectoires aléatoires : intégration de l'équation différentielle stochastique (par exemple, intégrale d'Itô numérique) produisant *graduellement*, pas à pas, une trajectoire.
    2. Échantillonnage aléatoire de trajectoires déterministes : le côté stochastique est réalisé par un choix aléatoire entre différentes trajectoires déterministes qui sont donc *complètement connues* dès le début de la simulation (pas facile à appliquer, sauf pour des courbes à *paramètres aléatoires*).

# Intégration numérique d'ED stochastique

- À la base d'une équation différentielle stochastique, l'intégrant suit un densité de probabilité connue.
  Ex.: modèle de la bande passante WiFi avec

$$\dot{p}(t) = \sigma(p(t))d\mathcal{P}(t)$$

- Pour réaliser l'intégration numérique, il faut discrétiser cette ED stochastique pour passer à une *équation aux différences* ; par exemple, pour un $\Delta t = t_{i+1} - t_i$ fixe, discrétiser $d\mathcal{P}(t)$ va utiliser une séquence de différences $\Delta_i$ telles que :

$$p_{i+1} - p_i = \sigma(p_i)\Delta_i$$

- Si on pose $p_0 = p(t_0)$, le calcul des valeurs discrétisées de l'intégrale $p_i = p(t_i), i = 0, \ldots$ devient :

$$p_i = p_0 + \sum_{j=0}^{i-1} \sigma(p_j)\Delta_j$$

# Comment calculer les différences ?

- Pour calculer la trajectoire, il faut obtenir des différences $\Delta_i$ *aléatoires* engendrées selon une densité de probabilité exprimée par $d\mathcal{P}(t)$ avec des générateurs de nombres pseudo-aléatoires.
  - Ex. (réseau WiFi) : nous avons supposé que la valeur absolue de l'intégrant suivait une loi exponentielle. En discrétisant, on obtient :

$$\Delta_i \sim \mathbf{Exp}[\lambda_i]$$

    - Pour un $\Delta t$ fixe, on peut supposer une exponentielle de moyenne fixe $(\forall i)\lambda_i \equiv \Lambda$, dépendant de $\Delta t$, mesurée expérimentalement.
    - Pour un $\Delta t$ variable, l'ampleur de la variation devrait croître en fonction de $\Delta t$ ; ici, on peut supposer que la moyenne $\lambda$ croît en fonction de $\Delta t$, par exemple $\lambda_i = (t_{i+1} - t_i) \times s$ où $s$ est un facteur d'échelle mesuré expérimentalement.
      (Fonctionne bien pour $\mathbf{Exp}[\lambda]$ ; pour une loi normale $\mathbf{N}[\mu, \sigma]$, ce serait plutôt la moyenne $\mu$ et/ou l'écart-type $\sigma$ qui croîtraient.)
- Une trajectoire sera alors obtenue grâce à l'équation précédente en y utilisant les nombres pseudo-aléatoires successifs générés.

# Plan

# Vers la simulation des systèmes hybrides

- Intégration pas-à-pas pour obtenir une trajectoire continue par morceaux, interrompue pour traiter deux types d'événements :
  1. ceux générés *explicitement* par leurs processus propres et traités comme dans la simulation par événements discrets mais en interrompant les calculs de la partie continue,
  2. ceux générés *implicitement* par l'évolution continue lors du franchissement de conditions de frontières (comme ceux déclenchés par l'état du modèle dans la simulation par événements discrets déjà vus, mais cette fois sur des variables continues).
- Principale difficulté : déterminer précisément le moment où les événements *déclenchés par l'évolution continue* se produisent.
  - Si on admet une tolérance $\varepsilon$, elle peut se répercuter par une modélisation trop infidèle du système.
  - Exemple : simulation hybride de la trajectoire d'une balle qui tombe sur une table et rebondit ; une tolérance $\varepsilon$ sur la position de la balle pour détecter la collision implique d'admettre que la balle pénètre d'une profondeur $\varepsilon$ *dans* la table...

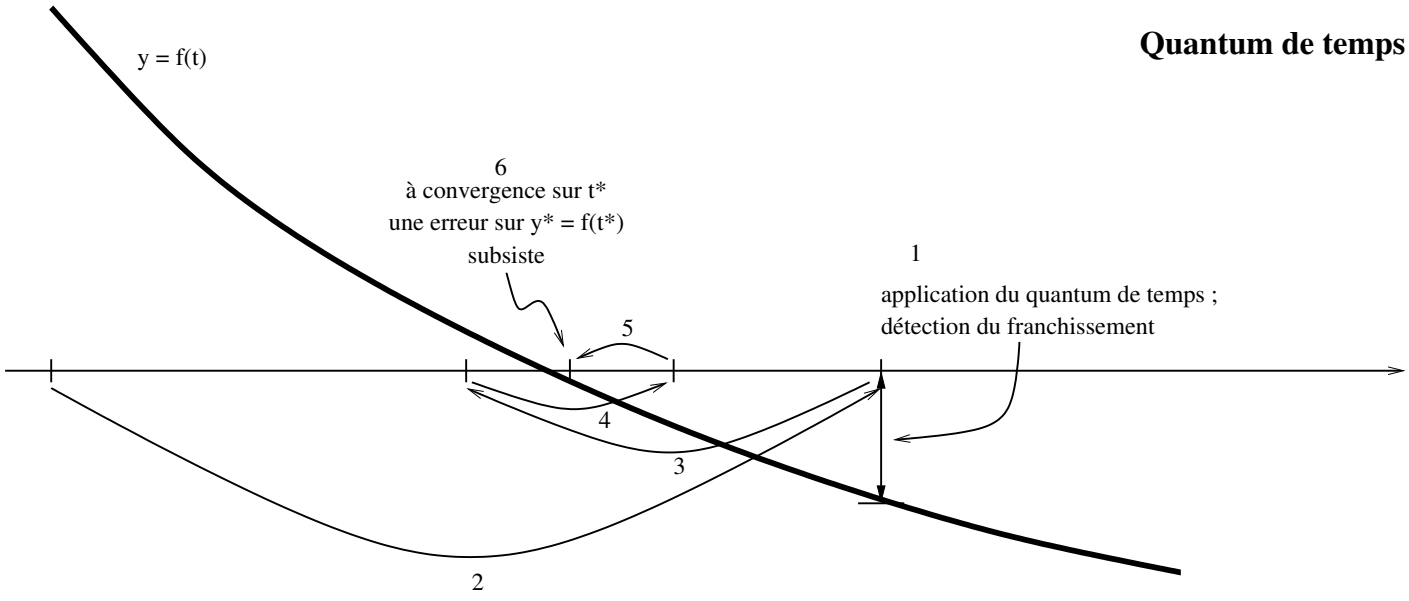
# Détermination des transitions en discrétisation du temps

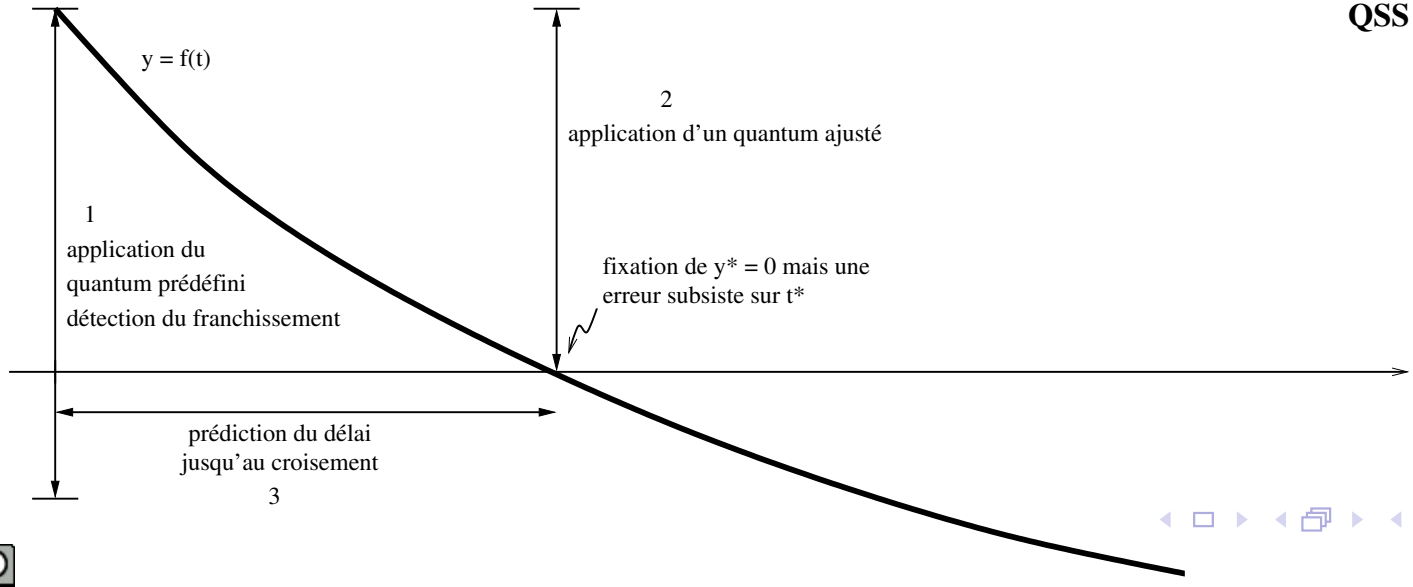
- Avec un pas d'intégration $\Delta t$, le simulateur discrétisant le temps saute en réalité de $t_i$ en $t_i + \Delta t$.
- Ceci produit une série d'instants qui n'ont aucune raison de correspondre aux instants d'occurrence des événements générés implicitement par l'évolution continue du système.
- Aucun pas d'intégration $\Delta t$ ne peut garantir une précision $\varepsilon$ donnée sur l'évolution des variables du modèle car les dérivées sont non-bornées.
  $\Rightarrow$ *Il faut rechercher l'instant d'occurrence plus précisément !*
- Se fait généralement en deux temps :
  - détection de la présence d'un franchissement de frontière entre $t_i$ et $t_{i+1}$ puis
  - recherche (dichotomique, ...) du point de franchissement précis à $\varepsilon$ près dans l'intervalle $]t_i, t_i + \Delta t[$.

# Détermination des transitions en discrétisation de l'état

- Constat : en discrétisation du temps, la recherche du moment d'occurrence du franchissement revient à se demander au temps $t_i$ dans combien de temps il va se produire.
  $\Rightarrow$ *C'est exactement ce que fait la discrétisation de l'état (QSS) !*
- Comment faire en discrétisation de l'état ?
  - Vérifier si le point de franchissement $\bar{y}$ se situe dans l'intervalle $y_i \pm \Delta y$ *i.e.*, $|y - \bar{y}| < \Delta y$.
  - Si c'est le cas, fixer temporairement le quantum $\bar{\Delta} y = |y - \bar{y}|$.
  - Déterminer dans combien de temps $\bar{\Delta} t$ le système atteindra $\bar{y} = y \pm \bar{\Delta} y$.
  - Avancer le système à $t_i + \bar{\Delta} t$ en fixant la nouvelle valeur de $y$ précisément à $\bar{y}$ (ainsi, plus d'erreur sur $y$).
- Il demeure possible de faire une erreur sur le délai $\bar{\Delta} t$ mais cela est souvent moins dommageable qu'une erreur sur l'état.
  - Par exemple, dans un jeu vidéo, il peut être plus choquant à l'oeil de détecter la pénétration d'une balle dans une table que de détecter un léger écart sur le moment du contact.

# Recherche de $t^* | f(t^*) = 0$ : quantum de temps versus QSS



# Événements discrets déclenchés par le temps

- Les événements discrets *e déclenchés par le temps* se produisent à des instants $t_e$ prévus avant leur occurrence :
  - événements déterministes $\Rightarrow$ $t_e$ est calculé au préalable ;
  - événements à occurrence stochastique $\Rightarrow$ *équivalents*, puisque la génération aléatoire se fait aussi au préalable.
- Les $t_e$ étant prévus, le moteur hybride consulte $t_e$ du prochain événement à chaque $t_i$ pour l'intercaler avant $t_i + \Delta t$.
  - Quantum de temps : si à $t_i$ on a $t < t_e \leq t_i + \Delta t$, alors
    - on ajuste temporairement $\Delta t$ à $\bar{\Delta} t = t_e - t_i$,
    - on avance la simulation continue jusqu'à $t_e = t_i + \bar{\Delta} t$ et
    - on exécute l'événement $e$, puis on reprend l'intégration.
  - En quantum d'état, on cherche le $\Delta t$ tel que $y_{i+1} = y_i \pm \Delta y$ à $t_{i+1} = t_i + \Delta t$ ; si $t_{i+1} > t_e$, il faut trouver un $\bar{\Delta} y$ tel que $y_{i+1} = y_i \pm \bar{\Delta} y$ à $t_e$ (par recherche dichotomique) pour exécuter $e$...

    Approche mixte : avancer en quantum d'état en vérifiant que $t_e > t_{i+1}$ et sinon ($t_e \leq t_{i+1}$), faire une étape en quantum de temps pour avancer exactement à $t_e$ pour exécuter $e$ puis reprendre l'intégration en quantum d'état.

# Récapitulons...

1. *Programmer* une simulation n'est pas si difficile, mais plus difficile de *bien modéliser*. Ne pas abuser des simplifications, en particulier lorsque des activités ont des interactions restant implicites dans le modèle.
   *Ex. : modélisation de la communication par réseau par des événements émission et réception, sans égard aux autres communications et à la congestion éventuelle du réseau.*
2. Dans le cas des modèles stochastiques, la disponibilité ou l'obtention des données réelles pour estimer des lois de probabilités justes est souvent l'une des plus grandes difficultés. La génération de valeurs aléatoires est aussi cruciale : évitez les mauvais générateurs des langages de programmation et préférez ceux des bibliothèques spécialisées (comme common-math).
3. Une seule exécution d'une simulation stochastique est insuffisante : il faut en faire beaucoup, selon un **plan d'expérimentation** bien défini et analyser scrupuleusement les résultats. L'aide d'un statisticien peut être utile.
4. L'intégration numérique selon l'approche de la **discrétisation de l'état** (*quantized state systems*) paraît la plus prometteuse pour produire des moteurs de **simulation hybride** précis et efficaces.

# Pour aller plus loin : sélection de lectures recommandées


- *Simulation Modeling & Analysis*, A.M. Law, 4è éd., McGraw Hill, 2007.
- *Guide to Modeling and Simulation of Systems of Systems*, B.P. Zeigler et H.S. Sarjoughian, Springer, 2013.
- *Numerical Analysis*, Richard Burden, Douglas Faires et Albert Reynolds, 2è édition, Prindle, Weber et Schmidt, 1981 (ou tout autre bon livre d'analyse numérique).
- B. Øksendal. Stochastic Differential Equations. Springer, 6th edition, 2013.
- *An algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations*, Desmond Higham, SIAM Reviews, 43, 3, pp. 525-546, 2001.