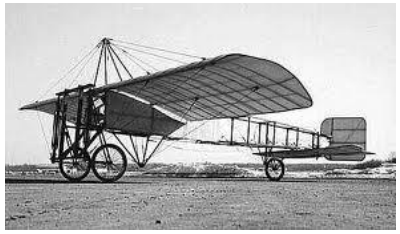
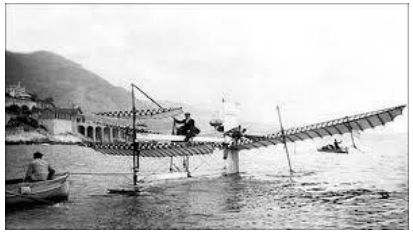
CentraleSupélec

Elements of Complex System Engineering

Antoine B. Rauzy
Department of Mechanical and Production Engineering (MTP)
Norwegian Science and Technology University (NTNU)
and
Chaire Blériot-Fabre
Centrale-Supélec, SAFRAN


CentraleSupélec

LECTURE 7.
DISCRETE EVENT SIMULATION

Notions:

- Discrete Event Systems
- Simulation Algorithms

CentraleSupélec

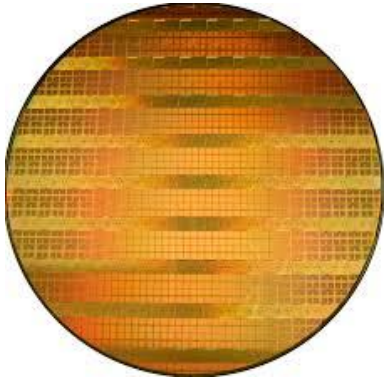
LECTURE 7. PART 1. INTRODUCTION

CentraleSupélec

Objective of this lecture

This lecture provides an introduction to **job-shop scheduling problems**. More importantly, it introduces the notion of **Discrete Event Systems,** and presents the principles of **Discrete Event Simulation**.

When considered from a sufficiently high abstraction level, many industrial systems can be described as discrete event systems. Discrete event simulation provides then an extremely powerful mean to analyze the system under study. Discrete event simulation is thus a fundamental tool of the toolbox of the system engineer.

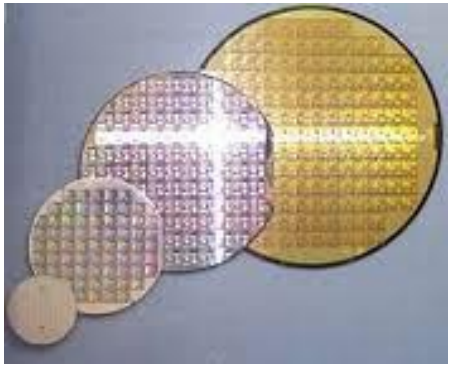


python

Case Study: Wafer Production Shop*

Wikipedia: “A wafer, also called a slice or substrate, is a thin slice of semiconductor material, such as a crystalline silicon, used in electronics for the fabrication of integrated circuits and in photovoltaics for conventional, wafer-based solar cells. The wafer serves as the substrate for microelectronic devices built in and over the wafer and undergoes many microfabrication process steps such as doping or ion implantation, etching, deposition of various materials, and photolithographic patterning. Finally the individual microcircuits are separated (dicing) and packaged.”

“Silicon wafers are available in a variety of diameters from 25.4 mm (1 inch) to 300 mm (11.8 inches). Semiconductor fabrication plants are defined by the diameter of wafers that they are tooled to produce. The diameter has gradually increased to improve throughout and reduce cost with the current state-of-the-art fab using 300 mm.”

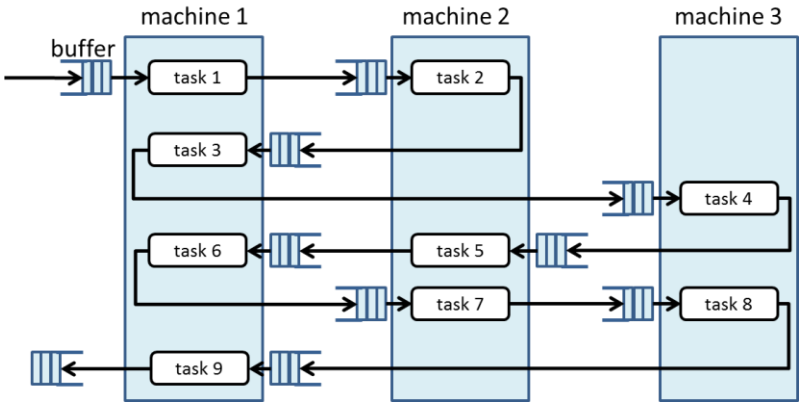


(*) This use case is inspired from:
Youngshin Park, Sooyoung Kim and Chi-Hyuck Jun. *Performance analysis of re-entrant flow shop with single job and batch machines using mean value analysis*. Production Planning and Control. Vol. 11, Num. 6. pp. 537-546. 2000.

Case Study: Wafer Production Shop (2)

Wafers are produced by **batches** of 20 to 50 wafers. To ensure the traceability of the process, batches are never split up.
The **production** consists in a series of **tasks**. Each task is performed by a **machine**. The same machine is used for several tasks (re-entrant flow), but it performs only one task at a time (of a batch of wafers).

Each machine has several **buffers**, one per task, in which it stores the batches waiting for a treatment. Each buffer corresponds to a task. Once a task performed on a batch, the batch is immediately discharged from the machine and loaded in the buffer of the next task.

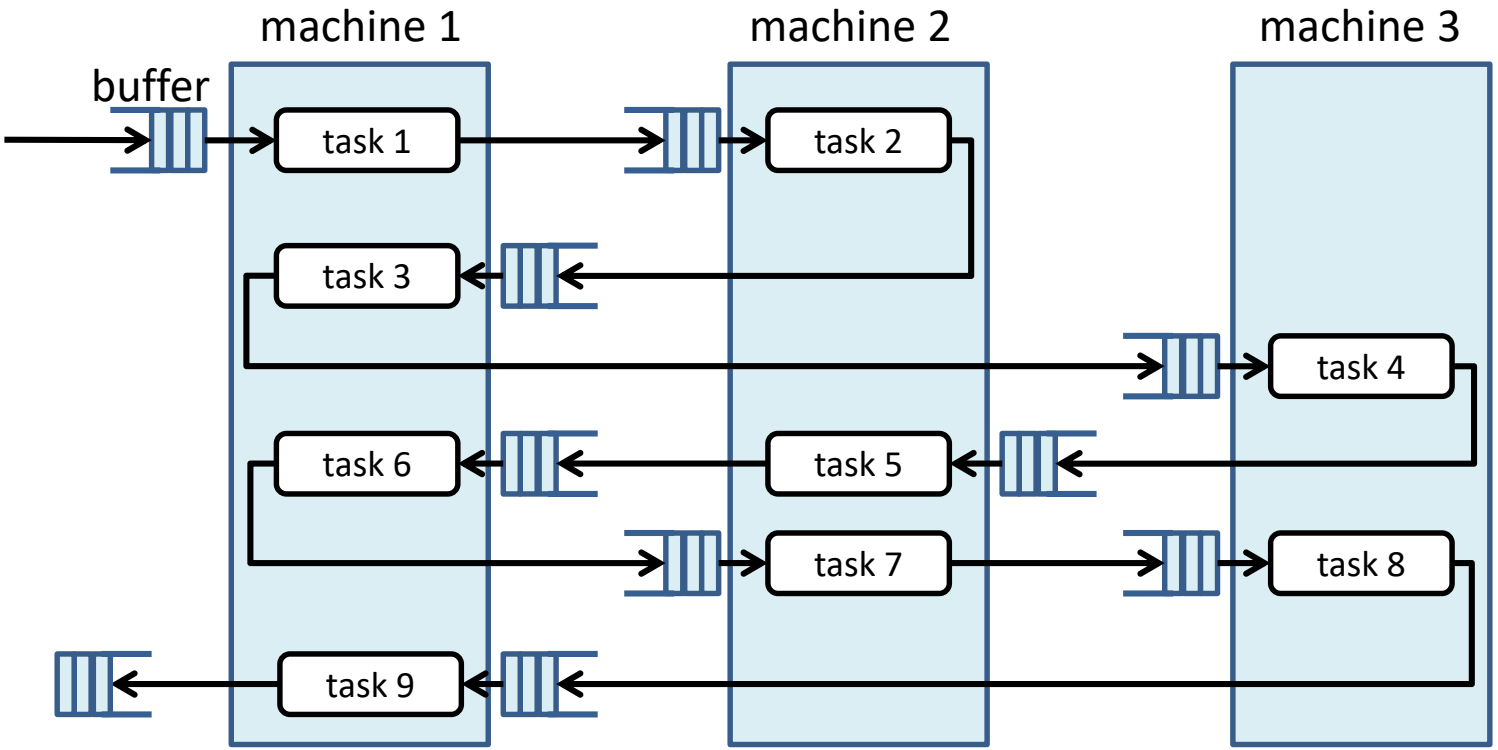


The **duration** of a task depends only on the time to perform the task on a single wafer and the time to load and unload a batch.

CentraleSupélec

Case Study: Wafer Production Shop (3)

Example:

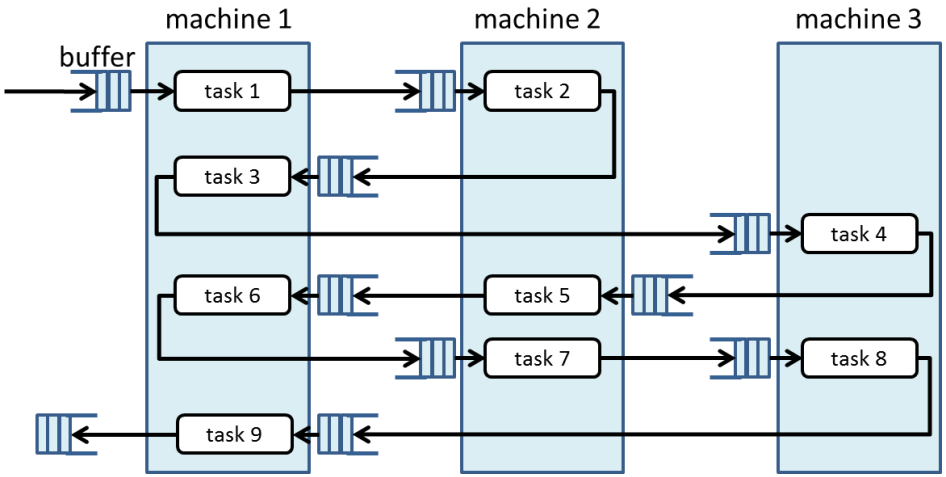


Task	1	2	3	4	5	6	7	8	9
Duration per wafer	0.5	3.5	1.2	3	0.8	0.5	1	1.9	0.3
Load/unload time	2	2	2	2	2	2	2	2	2

Case Study: Wafer Production Shop (4)

Machines (and their operation) are extremely costly. Therefore, it is of primary importance to **optimize** the production, i.e. to produce as many wafers as possible over a given time period. Concretely, this means tuning several **parameters**:

- The **size of the batches** of wafers (batches can be of different sizes);
- The **sizes of the buffers**;
- The **policy of each machine** (which waiting batch to treat first).



CentraleSupélec

What to Do?

It is of course possible to try to give a **mathematical formalization** to the problem so to get a (constraint) **optimization problem**, and then to try to solve this problem with some algorithm or meta-heuristics. In real life however, the fabrication process consists of about 60 different tasks performed on a dozen of machines.

The state space to explore is just gigantic and there will be probably no means to face this **combinatorial explosion**.

To analyze the problem a **good practical method** consists in creating a **discrete event model** that makes it possible to **simulate the behavior** of the shop. In this way, **virtual experiments** can be performed at a relatively low cost. The optimization can be partly computerized and partly done by hand, by means of a tries-and-errors approach.

This is the purpose of **discrete event simulation**.

CentraleSupélec

LECTURE 7. PART 2.
DISCRETE EVENT SYSTEMS

Finite State Automata (reminder)

A **finite state automaton** is a quadruple ⟨S, E, T, i⟩ where:

- S is a finite set of symbols called **states**;
- E is a finite set of symbols called **labels**; E is called the **alphabet** of the automaton.
- T is a subset of the Cartesian product S×E×S. Elements of T are called **transitions**. A transition (s,e,t) is often denoted: $e: s \rightarrow t$ or $s \xrightarrow{e} t$. The states s are called respectively the **source** and the **target** of the transition ;
- i is at state of S called the **initial state**.

The finite state automaton ⟨S, E, T, i⟩ is **deterministic**:

$$\forall\, s, t, t' \in S, \forall e \in E, s \xrightarrow{e} t \in T \text{ et } s \xrightarrow{e} t' \in T \Rightarrow t = t'$$

In other word, if each state s and each event e, there is at most one out-transition of s labelled by e.

Automata that describe system behaviors are not always deterministic.

Synchronized Products (reminder)

Let A_1: $\langle S_1, E_1, T_1, i_1\rangle$, … A_k: $\langle S_k, E_k, T_k, i_k\rangle$ be k finite state automata and let V be a set of synchronization vectors, i.e. a subset of the **Cartesian product** $E_1 \cup \{\varepsilon\} \times \ldots \times E_k \cup \{\varepsilon\}$ (the symbol ε represents the absence of transition).

The **synchronized product** of A_1,…, A_k by V is the automaton A: $\langle S, E, T, i\rangle$ such that:

- $S = S_1 \times \cdots \times S_k$
- $E = E_1 \cup \{\varepsilon\} \times \cdots \times E_k \cup \{\epsilon\}$
- $T = \left\{ \begin{array}{c} s = \langle s_1 \times \cdots \times s_k \rangle \in S \\ s \xrightarrow{e} t;\ t = \langle t_1 \times \cdots \times t_k \rangle \in S \\ e = \langle e_1 \times \cdots \times e_k \rangle \in E \end{array} \right.$ where for each i = 1..k $\left. \begin{array}{l} s_i \xrightarrow{e_i} t_i \in T_i \text{ si } e_i \neq \epsilon \\ s_i = t_i \text{ si } e_i = \varepsilon \end{array} \right\}$
- $i = \langle i_1 \times \cdots \times i_k \rangle$

The restriction of the automaton to its reachable states makes indeed fully sense with the notion of synchronized product: the synchronized product is built by means of a fixpoint mechanism from its initial state.

CentraleSupélec

Discrete Event Systems

A **Discrete Event System** (DES) is 4-tuple $\langle V, E, T, i \rangle$ where :

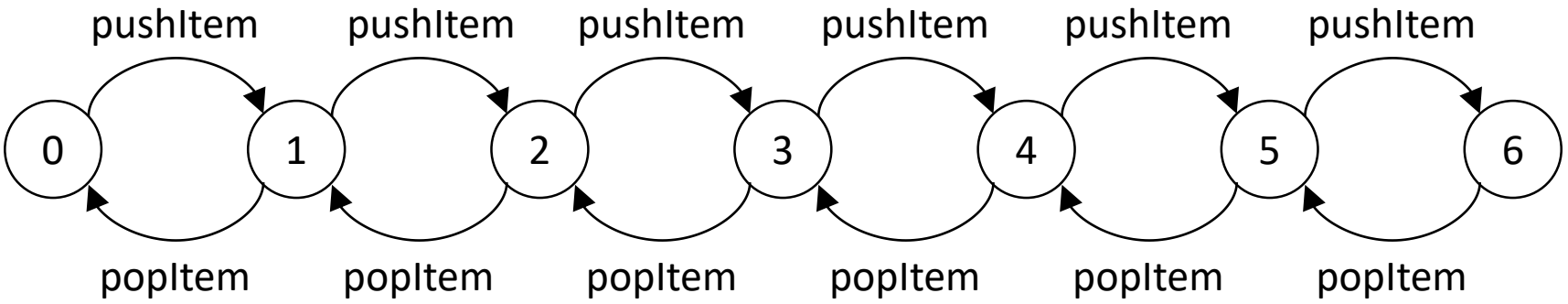
- V is a set of **variables** describing the **state** of the system. In general, variables of V take their values into **finite domains**;
- E is a set of **events**;
- T is a set of **transitions**. A transition is a triple $\langle G, e, A \rangle$, denoted as $e: G \rightarrow A$, where:
  - G is a Boolean function over V, i.e. a mechanism that given a value of the variables returns true or false. G is called the **guard** of the transition.
  - e is an event of E.
  - A is a function from V to V, i.e. a mechanism that given a value of variables returns a new value for these variables. A is called the **action** of the transition.
- i is the **initial assignment** of variables.

A transition $e: G \rightarrow A$ is **fireable** in the state σ, i.e. for the variable assignment σ, if $G(\sigma)$ = true.

Example: a Simple Buffer

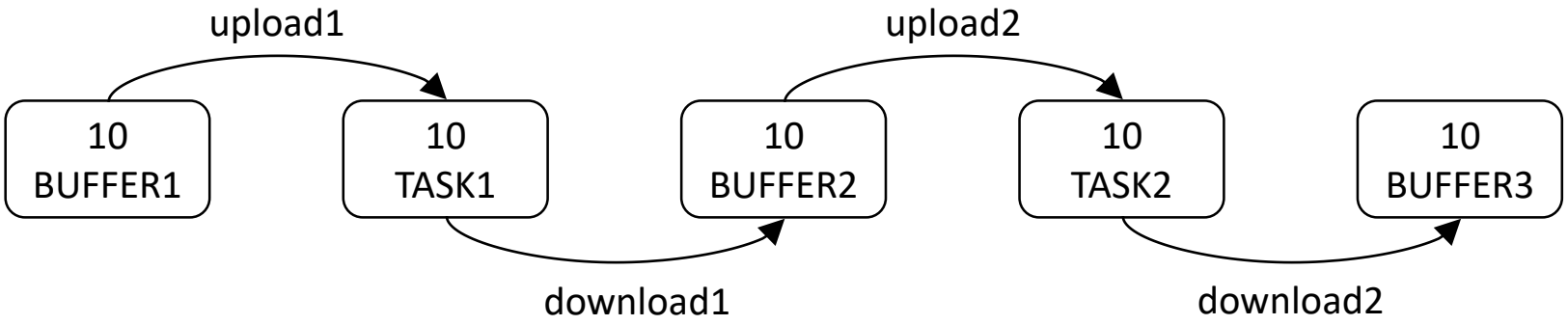
- V = { numberOfItems }, dom(numberOfItems) = [0, 6]
- E = { pushItem, popItem }
- T = {
  pushItem: numberOfItems<6 → numberOfItems := numberOfItems+1
  popItem: numberOfItems>0 → numberOfItems := numberOfItems-1
  }
- ι = [numberOfItems = 0]

The above **discrete event system** describes **implicitly** the following **finite state automaton**.

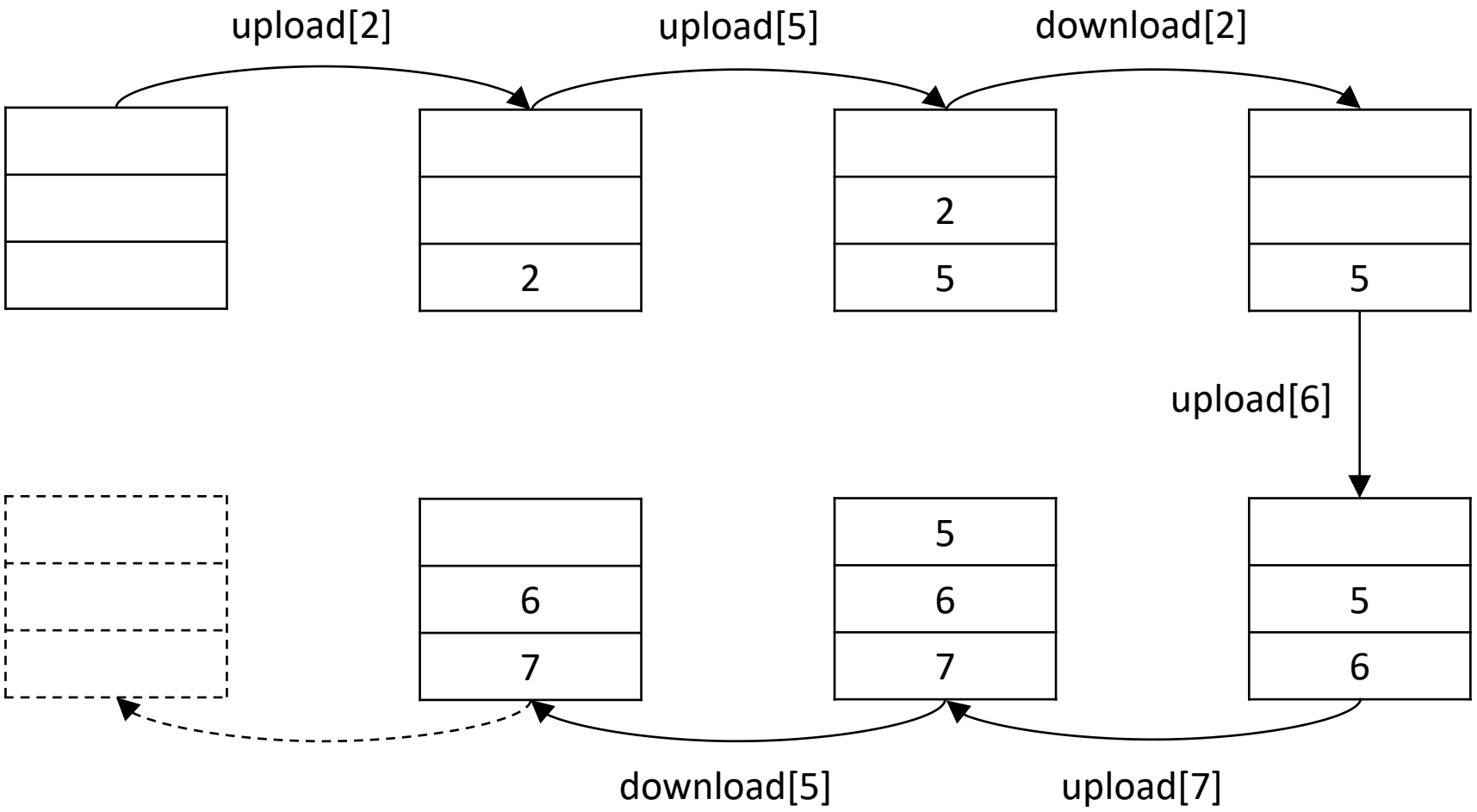


Example: a 10 wafers Batch

- V = { numberOfWafers, status}
  dom(numberOfWafers) = {10}
  dom(status) = { BUFFER1, TASK1, BUFFER2, TASK2, BUFFER3 }
- E = { upload1, download1, upload2, download2 }
- T = {
  upload1: status= BUFFER1 → status := TASK1
  download1 : status= TASK1 → status := BUFFER2
  upload2: status= BUFFER2 → status := TASK2
  download2 : status= TASK2 → status := BUFFER3
  }
- ι = [numberOfWafers = 10, status = BUFFER1]

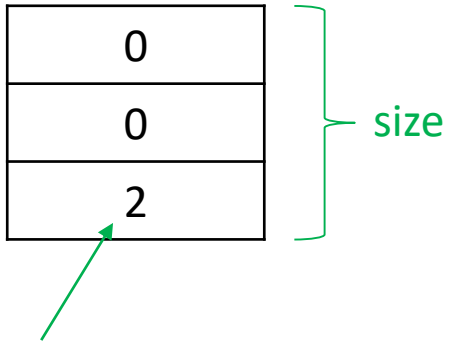


Example: a Queue



Example: a Queue

- parameters: size, numberOfItems
- V = {top} U { level[i] ; i ∈ [0, size-1] }
  dom(top) = [0, size-1]
  dom(level[i]) = [0, numberOfItems]
- E = { upload[i], download[i]; i ∈ [1, numberOfItems] }
- T = {
  upload[i]: top<size →
  for l=top downto 1: level[l] := level[l-1]
  level[0] := i
  top := top+1
  download[i]: top>0 and level[top-1]=i →
  level[top-1] = 0
  top := top-1
  for i=1,… numberOfItems
  }
- ι = [top= 0, for l=0 to size-1: level[l]:=0]



Composition

Let A: $\langle V_A, E_A, T_A, \iota_A \rangle$ and B: $\langle V_B, E_B, T_B, \iota_B \rangle$ be two discrete event systems built on distinct sets of variables and events.

The **composition** (or **product**) of A and B, denoted $A \otimes B$, is the discrete event system $\langle V, E, T, \iota \rangle$ defined as follows.

- $V = V_A \cup V_B$
- $E = E_A \cup E_B$
- $T = T_A \cup T_B$
- $\iota = \iota_A \circ \iota_B$

To avoid collisions in the name of variables and events, it is convenient to prefix all of the variables and events of A by "A." and all of the variables and events of B by "B.".

The product defined above is commutative and associative. Therefore it can be extended to any number of discrete event systems.

Synchronization and Hiding

Event **synchronization** is possible (and useful). It is an internal operation. It consists in defining a transition as the synchronization of two or more transitions.

Let $\langle V, E, T, \iota \rangle$ be a DES, let $e_1: G_1 \rightarrow A_1$ and $e_2: G_2 \rightarrow A_2$ be two transitions of T and finally let e be an event of e. We can introduce the new transition e: e_1 & e_2 defined as follows.

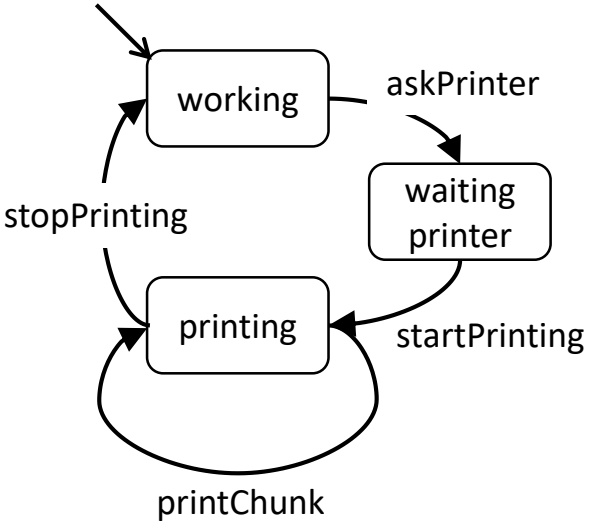
e: G_1 and $G_2 \rightarrow A_1$ o A_2;

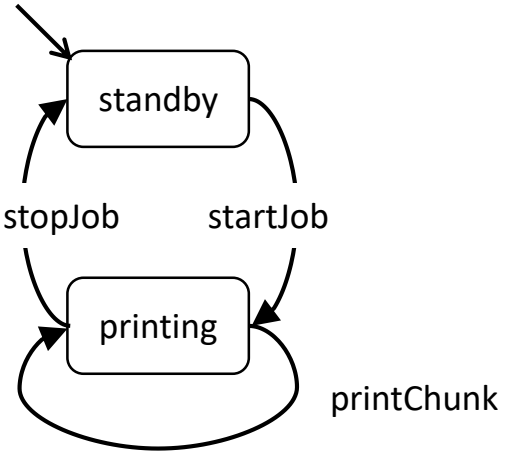
The synchronization of two events is associative, it can be thus extended to any number of events.

Hiding is sometimes convenient when building discrete event systems by composing smaller ones. It consists is forbidding a event to occur individually (but this event may have been synchronized before).

Network (reminder)

A slightly more realistic model



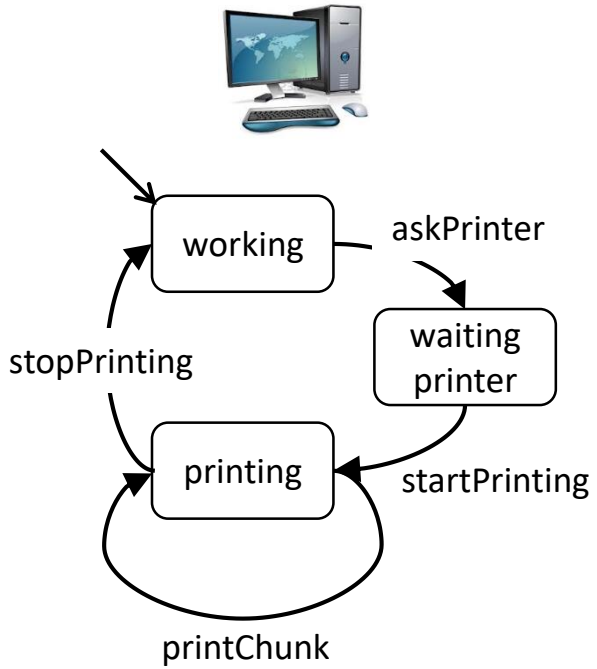


	computer 1	computer 2	printer
askPrinter1	askPrinter		
askPrinter2		askPrinter	
startPrinting1	startPrinting		startJob
startPrinting2		startPrinting	startJob
stopPrinting1	stopPrinting		stopJob
stopPrinting2		stopPrinting	stopJob
printChunk1	printChunk		printChunk
printChunk2		printChunk	printChunk

DES for Computers

Computer:

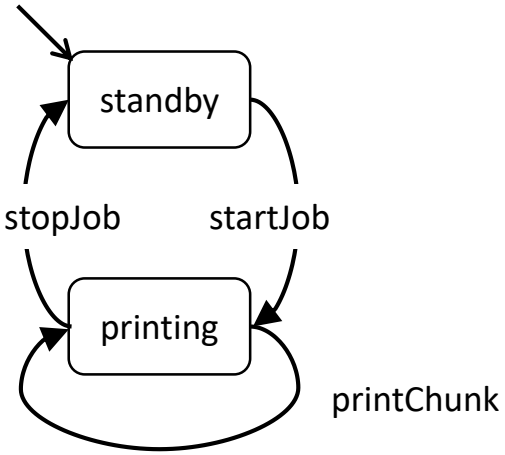
- V = { state }
  dom(state) = { WORKING, WAITING, PRINTING}
- E = { askPrinter, startPrinting, printChunck, stopPrinting }
- T = {
  askPrinter: state=WORKING → state := WAITING
  startPrinting: state=WAITING → state := PRINTING
  printChunck: state=PRINTING → state := PRINTING
  stopPrinting: state= PRINTING → state := WORKING
  }
- ι = [state=WORKING]



DES for Printers

Printer:

- V = { state }
  dom(state) = { STANDBY, PRINTING}
- E = {startJob, printChunck, stopJob }
- T = {
  startJob: state=STANDBY $\rightarrow$ state := PRINTING
  printChunck: state=PRINTING $\rightarrow$ state := PRINTING
  stopJob: state= PRINTING $\rightarrow$ state := STANDBY
  }
- ι = [state=STANDBY]



DES for the Network

Network:

- V = { C1.state, C2.state, P.state }
  dom(C1.state) = dom(C2.state) = { WORKING, WAITING, PRINTING}
  dom(P.state) = { STANDBY, PRINTING}
- E = { askPrinter1, startPrinting1, printChunck1, stopPrinting1,
  askPrinter2, startPrinting2, printChunck2, stopPrinting2 }
- T = {
  askPrinter1: C1.state=WORKING $\rightarrow$ C1.state := WAITING
  startPrinting1: C1.state=WAITING and P.state=STANDBY $\rightarrow$
  C1.state := PRINTING, P.state := PRINTING

  …
  stopPrinting2: C2.state= PRINTING and P.state=STANDBY $\rightarrow$
  C2.state := WORKING, P.state := STANDBY
  }
- ι = [C1.state=WORKING, C2.state=WORKING, P.state=STANDBY]

Synchronization & Hiding

Synchronization:

C1.startPrinting: C1.state=WAITING → C1.state := PRINTING
&
P.startPrinting: P.state=STANDBY → P.state := PRINTING
=
startPrinting1: C1.state=WAITING and P.state=STANDBY →
C1.state := PRINTING, P.state := PRINTING

Hiding:

C1.startPrinting and P.startPrinting cease to exist individually

CentraleSupélec

Reachability Graph

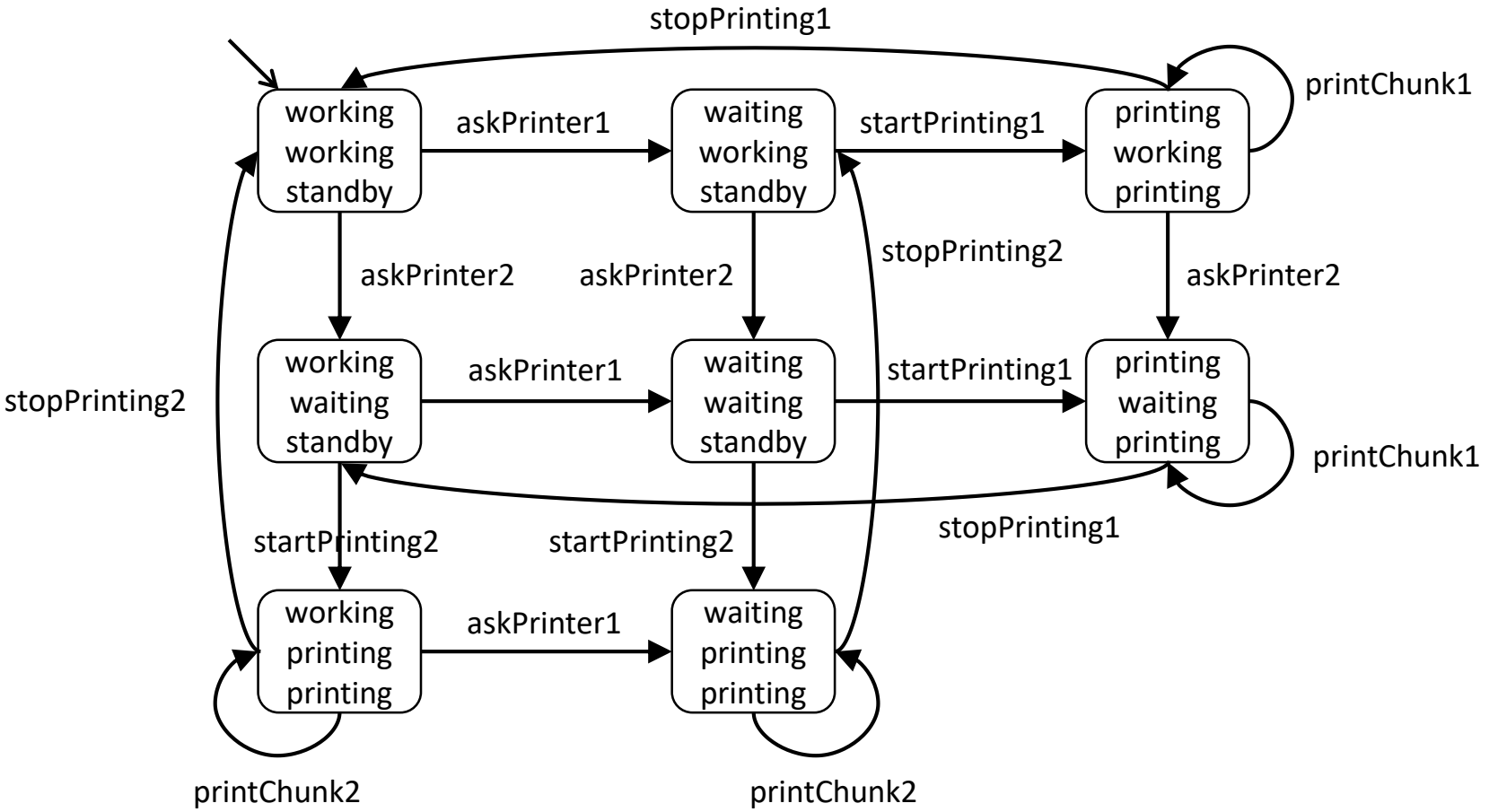
The **reachability graph** of a discrete event system $\langle V, E, T, \iota \rangle$ is a **Kripke structure** $\langle \Sigma, \Theta \rangle$ built as follows.

- The initial assignment ι belongs to Σ.
- If a variable assignment σ belongs to Σ, and there is a transition e: G $\rightarrow$ A of T such that G(σ) = true, then:
  - τ = A(σ) belongs to Σ;
  - e: s $\rightarrow$ t belongs to T.

The DES $\langle V, E, T, \iota \rangle$ is therefore an **implicit representation** of the Kripke structure $\langle \Sigma, \Theta \rangle$.

For most of the industrial size models, the Kripke structure cannot be built because it is much too big.

Reachability Graph of the Network



CentraleSupélec

Runs

An alternative way of defining the semantics of discrete event systems consists in using the notion of run.

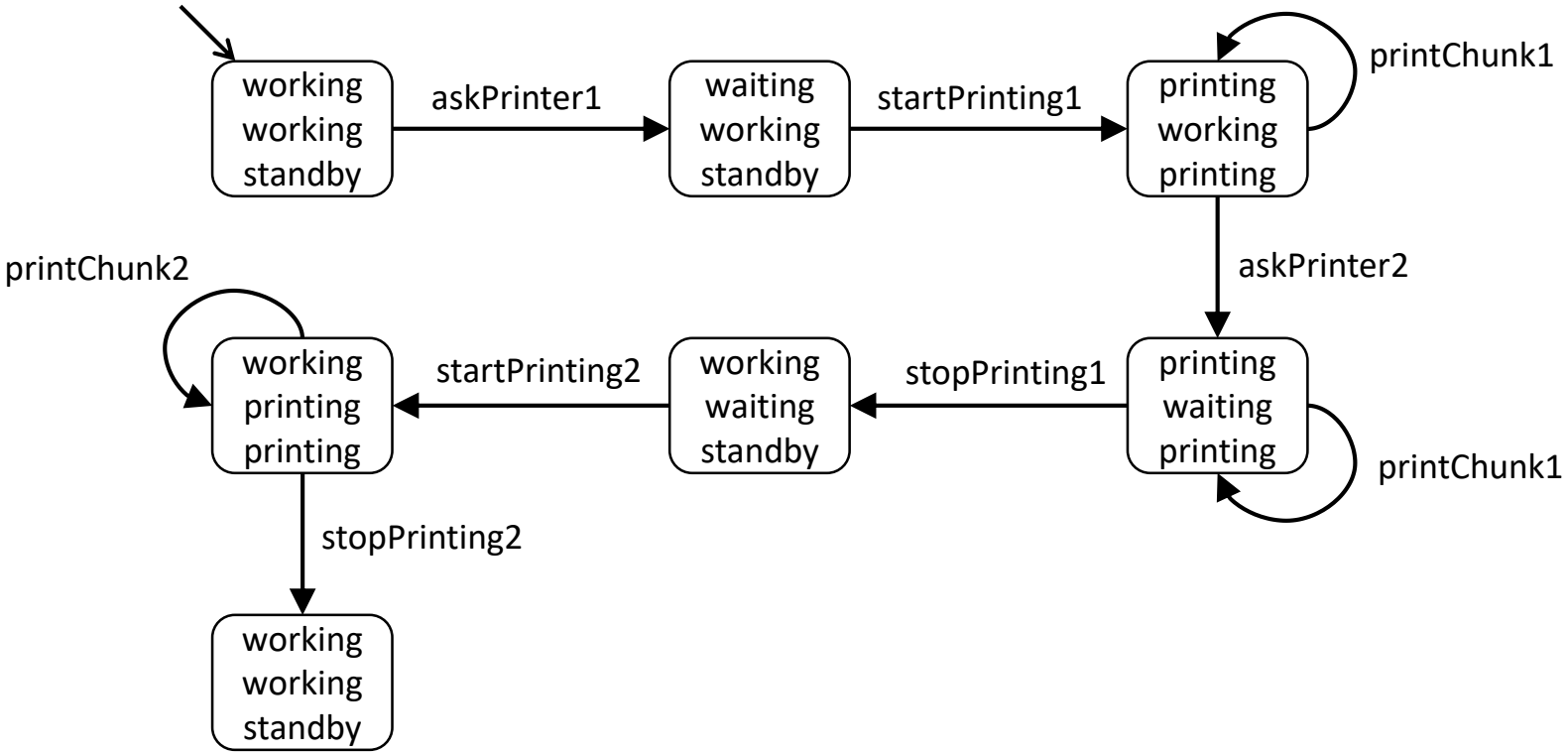
Let M: $\langle V, E, T, \iota \rangle$ be a discrete event system. A **run** of M is a finite alternated sequence of variable assignments and events $\langle \sigma_0, e_1, \sigma_1, e_2, \ldots \sigma_n \rangle$.

The set R_M of runs of M is the smallest set of runs such that:

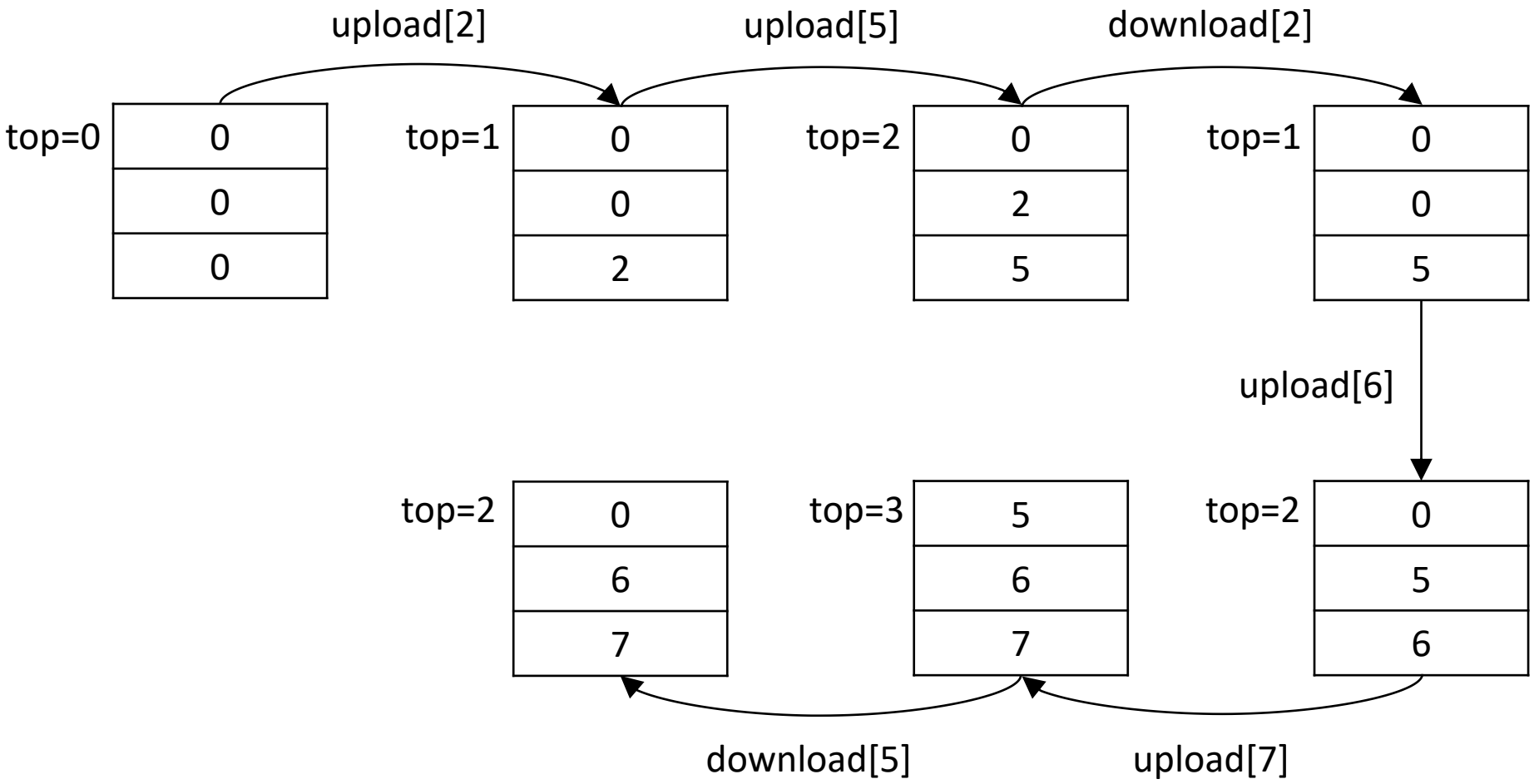
- $\langle \iota \rangle$ belongs to R_M. It is the only empty run. It starts and ends in ι.
- If $\langle \iota, \ldots, \sigma \rangle$ is a run of R_M ending by the state (variable assignment) σ, then for all transitions e: G $\rightarrow$ A of T such that G(σ) = true then
  - $\langle \iota, \ldots, \sigma, e, A(\sigma) \rangle$ is a run of R_M ending in the state A(σ).

Note that there may be an infinite number of runs of M even though its reachability graph is finite.

A Run of the Network



A Run of the Queue

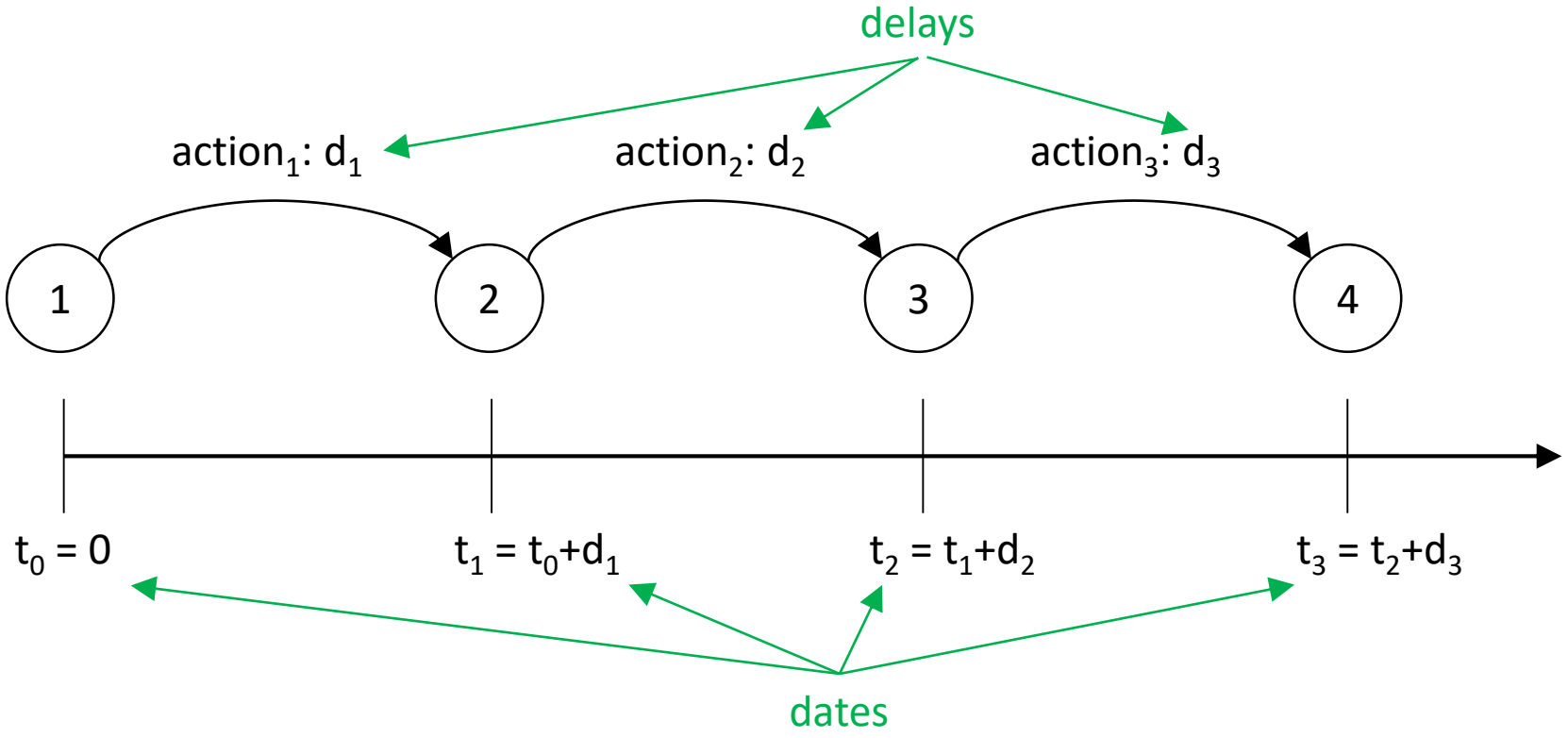


CentraleSupélec

LECTURE 7. PART 3.
TIMED DISCRETE EVENT SYSTEMS

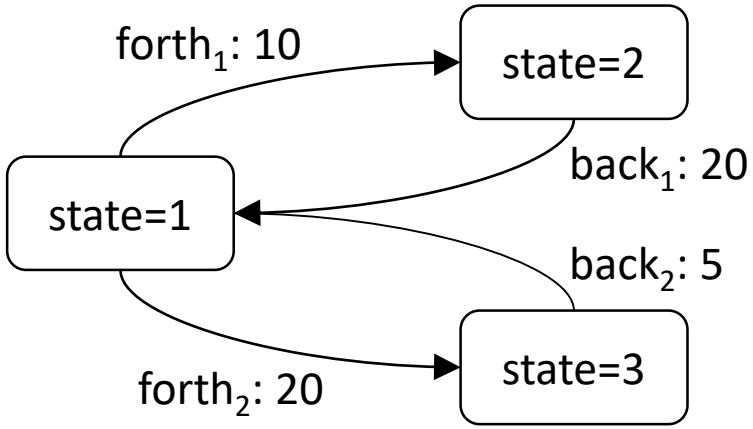
Timed Transitions

To model accurately systems like the wafer production plant, we need to take into account the time necessary to perform actions such as the treatment of a batch. The idea is therefore to introduce **timed transitions**.

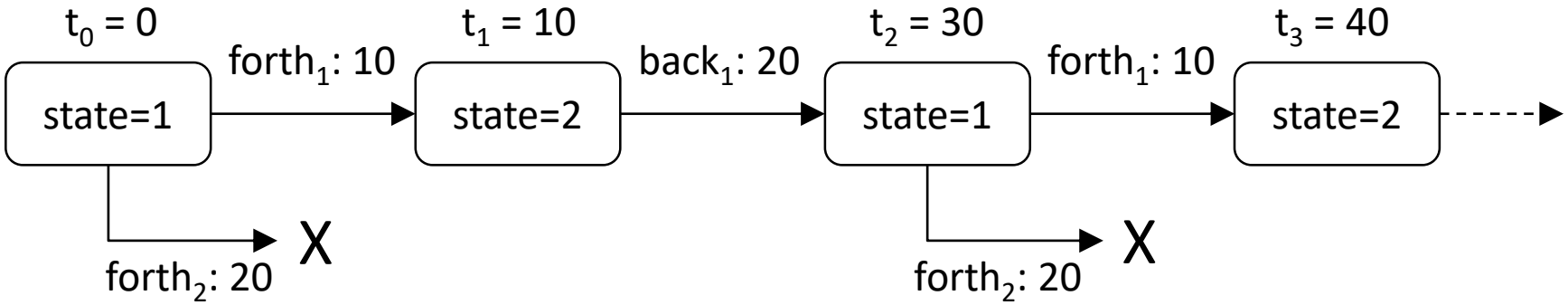


Changes in Operational Semantics

The introduction of delays changes the operational semantics of discrete event systems.

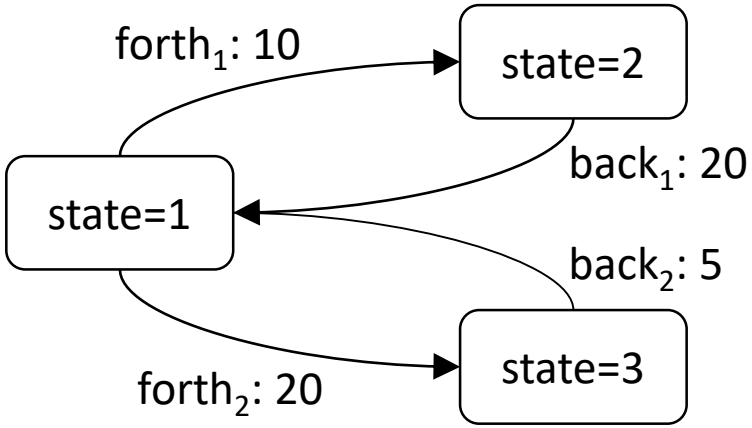


With the un-timed semantics, both forth$_1$ and forth$_2$ can be fired when the system in state 1.
With the timed semantics, the transition forth$_2$ will never be fired: the firing of transition forth$_1$ falsifies the guard of the transition forth$_2$, and therefore makes it un-fireable.



Schedule

A **schedule** is the list of fireable transitions in a given state and at a given date, together with a firing dates for each of the transitions.



schedule in state=1 at $t_0 = 0$:

- forth$_1$, t_0+10 = 10
- forth$_2$, t_0+20 = 20

The transition to be fired is the one with the smallest date, here forth$_1$

Firing a transition consists in:

1) Performing the action of the transition.
2) Increase the current date by the delay of the transition.
3) Remove from the schedule all transitions that are not fireable anymore.
4) Add to the schedule all transitions that became fireable.

Schedule in state=2 at t_1=10 (after firing the transition forth$_1$):

- ~~forth$_1$, t_0+10 = 10~~
- ~~forth$_2$, t_0+20 = 20~~
- back$_1$, t_1+20 = 30

- forth1 and forth$_2$ are removed
- back$_1$ is added

Timed Runs

Let M: $\langle V, E, T, \iota \rangle$ be a timed discrete event system. A **timed run** of M is a finite alternated sequence $\langle(\sigma_{0,}, s_0), (e_{1,},d_1), (\sigma_1, s_1), (e_{2,},d_2),\ldots (\sigma_{n,}, s_n)\rangle$ where the σ_i's are variable assignments, the s_i's are schedules, the e_i's are events and the d_i's are delays.

The set R_M of runs of M is the smallest set of runs such that:

- $\langle \iota, s_0\rangle$ belongs to R_M. It is the only empty run. It starts and ends in ι. s_0 is the schedule made of fireable transitions at t=0.
- If $\langle(\iota, s_0),\ldots , (\sigma_{n,}, s_n)\rangle$ is a run of R_M ending by the state (variable assignment) σ with the schedule s, and e: G $\rightarrow$ A of T is a transition of delay d and such that G(σ) is true then $\langle(\iota, s_0),\ldots, (\sigma_{n,}, s_n), (e_{,}d), (\sigma_{n+1,}, s_{n+1})\rangle$ is a run of R_M, if:
  - $\sigma_{n+1} = A(\sigma_n)$
  - s_{n+1} is the schedule obtained by updating s_n with the firing of e: G $\rightarrow$ A

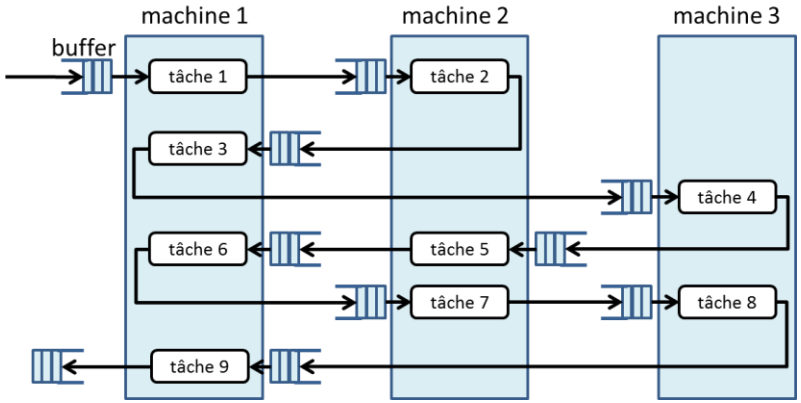
CentraleSupélec

LECTURE 7. PART 4.
APPLICATION TO THE USE CASE

Events and Transitions of the Use Case

In the use case, there are 4 types of events:

- Load a batch into a buffer;
- Unload a batch from a buffer;
- Start task on a machine and a batch;
- End task on a machine and a batch.



We can assume that:

- Unloading a batch from a buffer is immediately followed by starting a task on the corresponding machine. The two events can thus be merged into a single event ”StartTask”.
- The end of a task on a batch is immediately followed by loading the batch into the next buffer. The two events can thus be merged into a single event “EndTask”. Loading a batch into the first buffer is a special event “LoadBatch”.

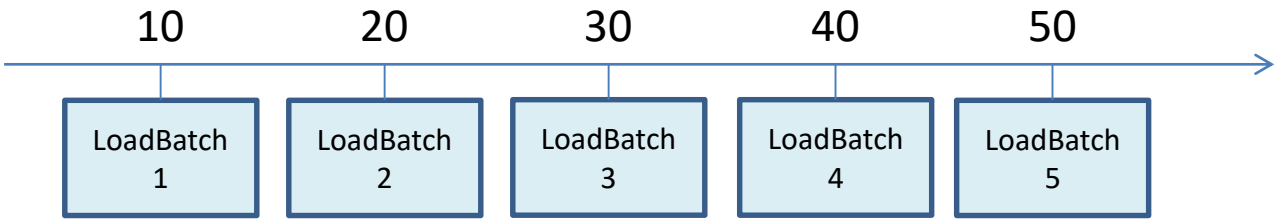
Simulation (1)

Assume we want to produce 5 batches of 10 wafers each (numbered from 1 to 5).

task	1	2	3	4	5	6	7	8	9
duration (for one batch)	5	35	12	30	8	5	10	19	3

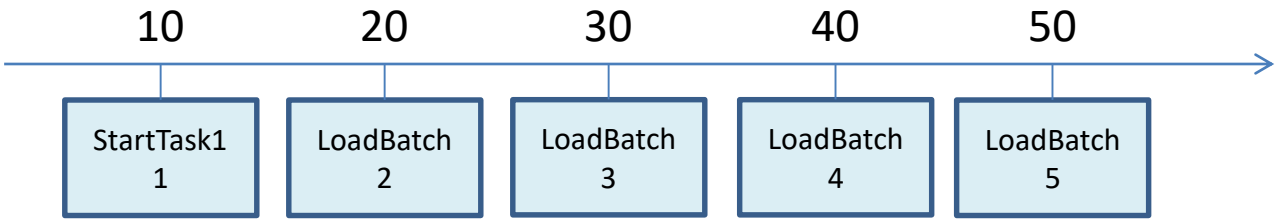
Assume moreover that we load the production chain with batches successively at dates 10, 20, 30, 40 and 50. The initial schedule is thus the following:

step 1

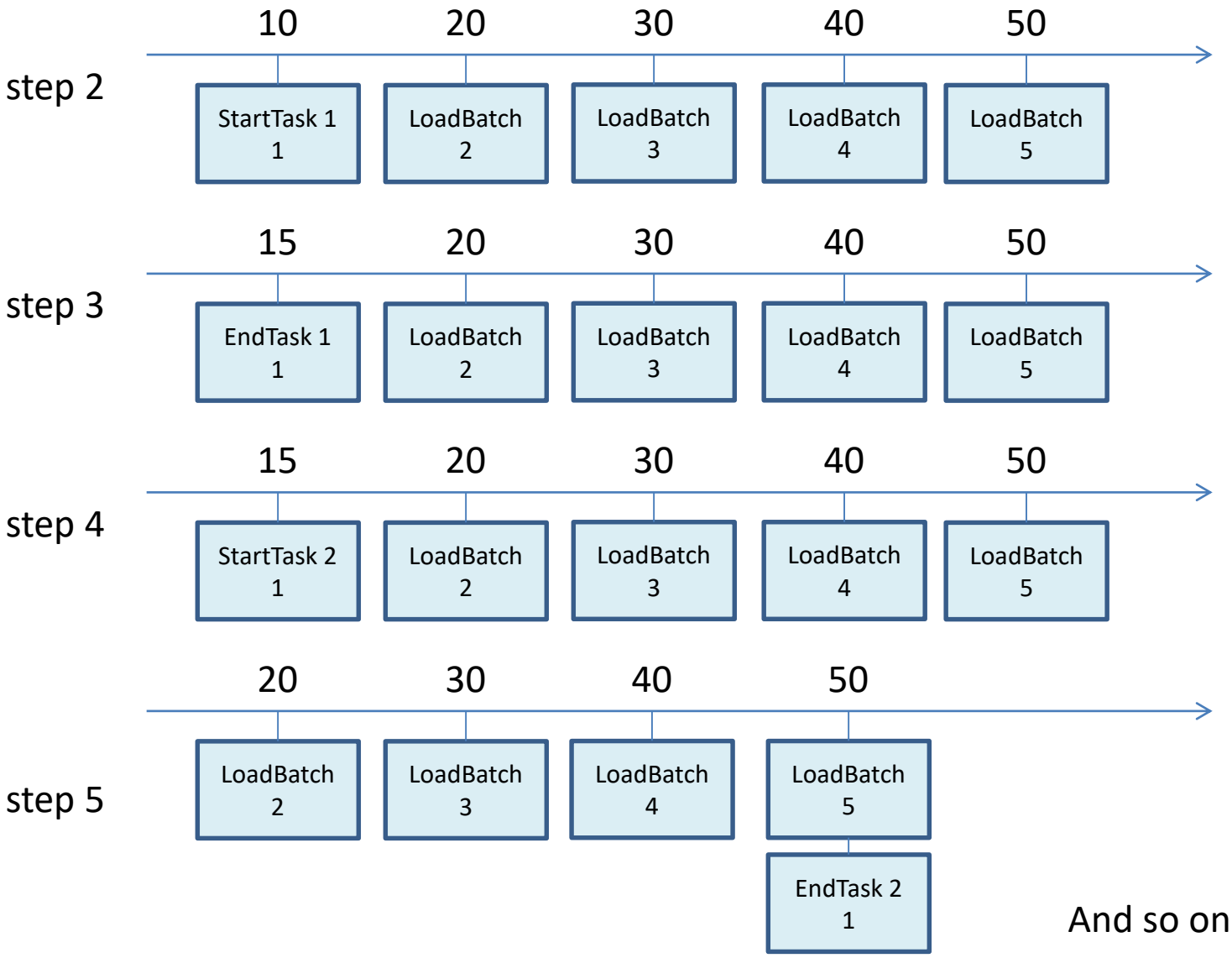


And the simulation starts:

step 2



Simulation (2)



And so on....

CentraleSupélec

Production Optimization

We can use the discrete event simulation as the base brick for production optimization.

Given a number of wafers to be produced, we can try to find the best combination of:

- Policies of the machines (which task has the priority in which situation);
- Number and size of batches;
- Dates of introduction of batches into the production chain.

The best combination is the one that makes it possible to complete the production within the shortest time.

The search for the optimal combination can be done manually “try and test” approach or by means of some optimization algorithm.

In any case, the state space to be look at is so huge that no exhaustive search is possible. Moreover, this use case shows high irregularities and non-linearity.

CentraleSupélec

LECTURE 7. PART 5.
WRAP-UP & ASSIGNMENT

CentraleSupélec

Wrap-Up

- **Discrete Event Simulation** is one of the base tool of complex systems engineering. It makes it possible to study the behavior of these systems even when no “analytical” tool is available.
- Discrete event simulation relies on the notion of **timed automaton**. Many different types of automata may be used as a support formalism, but the basic principles are those provided in this lecture.

CentraleSupélec

Assignment

The objective of this assignment is:

1. To learn about workshop scheduling problems;
2. To get more familiar with discrete event simulator;
3. To use Python.

You shall modify the program “myjobshop.py” and make some experiments with.

Question 1: Look at the program. Try to understand it. Run it. Comment.

Question 2: What is the sequence of events to produce 2 batches of 50 wafers. Same question with 3 batches.

Question 3: What is the best strategy to produce 1000 wafers. What are the occupation rates of the machines?

Question 4: The method “SelectBatchAndTaskToStart” of the class “WorkStation” defines a strategy to choose the next task to perform. Is there a better strategy?

Recommend Readings

Recommended books on discrete event systems:

Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer US, 2008. 978-0-387-33332-8

CentraleSupélec

Louis Charles Joseph Blériot (1872 -1936) is an airplane designer and one of the pioneer pilot of French aviation. He has been the first to cross the channel on July the 25th onboard of the Blériot XI.
He graduated from Ecole Centrale de Paris

Henri Marie Léonce Fabre (1882 -1984) is a French engineer and pilot. He invented the seaplane in 1910.
He graduated from Supélec.