

Parallelism

Master 1 International



Andrea G. B. Tettamanzi

Université de Nice Sophia Antipolis

Département Informatique

andrea.tettamanzi@unice.fr

Lecture 3, Part b

Describing Concurrent and Parallel Algorithms

Table of Contents

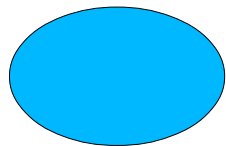
- “Informal” Modeling Tools (Formal tools will be presented later):
 - Data-Flow Diagrams
 - Synchronous Data Flow (SDF) Graphs
 - Activity Diagrams
 - Petri Nets
 - Actor Event Diagrams
 - Algorithmic Skeletons
- Methodical Design of Parallel Algorithms

Data Flow

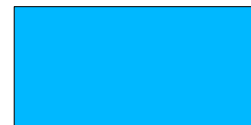
- A model of computation
- Computing nodes execute whenever input data is available
- Nodes are connected by “data paths”
- A special class of algorithms can be modeled as synchronous data flow (SDF) graphs
- A data flow graph is based on data dependencies
- A node can in turn represent another data flow graph (abstraction)
- A node may be implemented as a sequential program

Data-Flow Diagrams

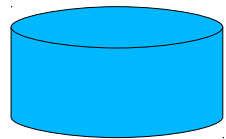
- A time-honored modeling tool in Computer Science
- Graphical representation of the flow of data through an information system
- A DFD shows tasks with their inputs and outputs
- Four graphical elements:



Function



Input/Output



File/Database

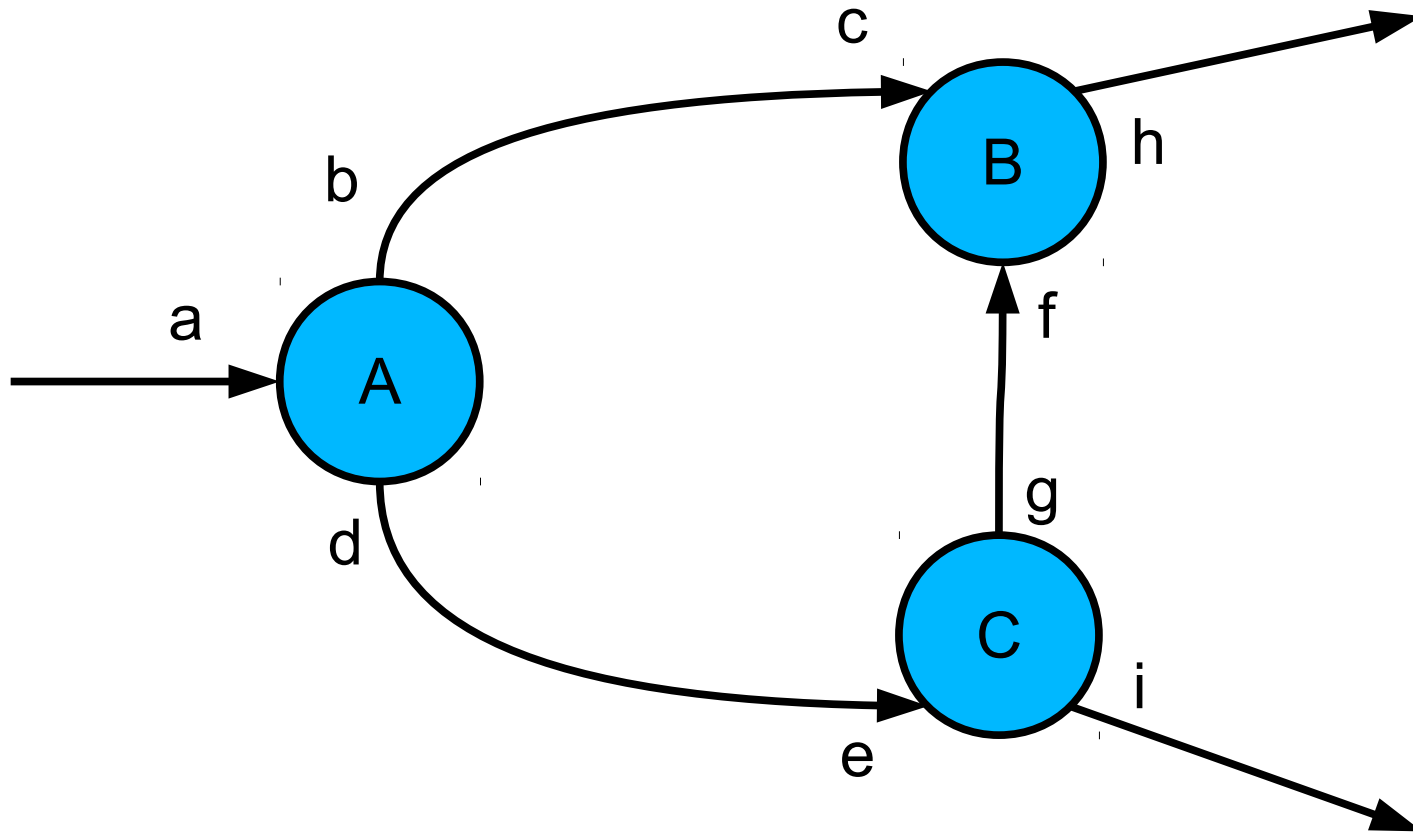


Flow

Synchronous Data-Flow Graphs

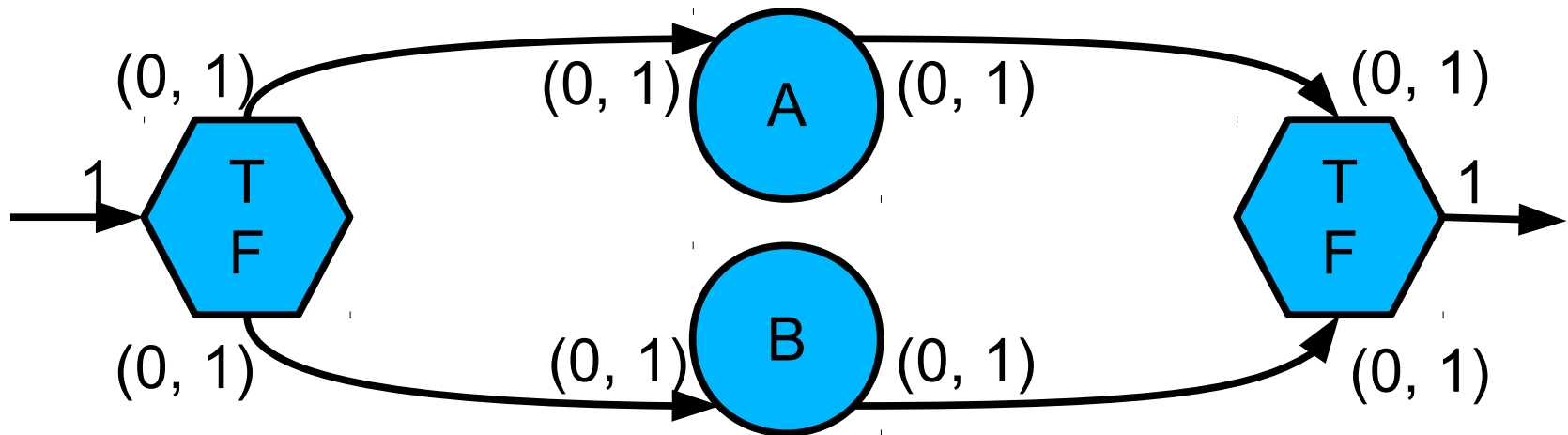
- A special class of algorithms can be modeled as synchronous data flow (SDF) graphs for which efficient implementation methods exist
- A SDF graph is a special case of a data flow
- Data are assumed to consist of “tokens”
- A node is said to be *synchronous* if the number of input tokens that are consumed on each input and the number of output tokens that are produced on each output can be specified a priori.
- Nodes are free from side effects
- Nodes may have a state, but this state does *not* influence the number of tokens consumed and produced in each cycle
- Originally proposed for signal processing

SDF Graphs



SDF Graphs: Control Nodes

- The “switch” and “select” functions cannot be represented as an SDF node because the number of tokens produced (switch) or consumed (select) cannot be decided a priori
- However, the coarse-grained data flow can still be represented as an SDF graph



SDF Graphs: Properties

- Compared to data flow, synchronous data flow has the following nice properties:
 - In contrast to data flow graphs scheduling does not have to be done at runtime, but can be done at compile time
 - If a periodic admissible parallel schedule can be found, the memory for buffers is bounded
 - Mathematical methods to derive a schedule exist
- Compiling an SDF graph:
 - find a periodic admissible parallel schedule (PAPS)

Activity Diagrams

- They are essentially an extension of flow charts
- Part of the UML
- Model the dynamical aspects of a system
- Describe the flow of control from an activity (= task) to another
- Activity
 - Sequence of atomic computations
 - Ends up in an action
- Transition
 - Transfer of control from an activity to another
- Object
 - Result of some activities

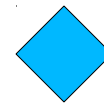
Activity Diagrams: Graphical Elements

● **Initial state**

⦿ **Final state**

Description

Activity



Decision



Transition



Fork/Join

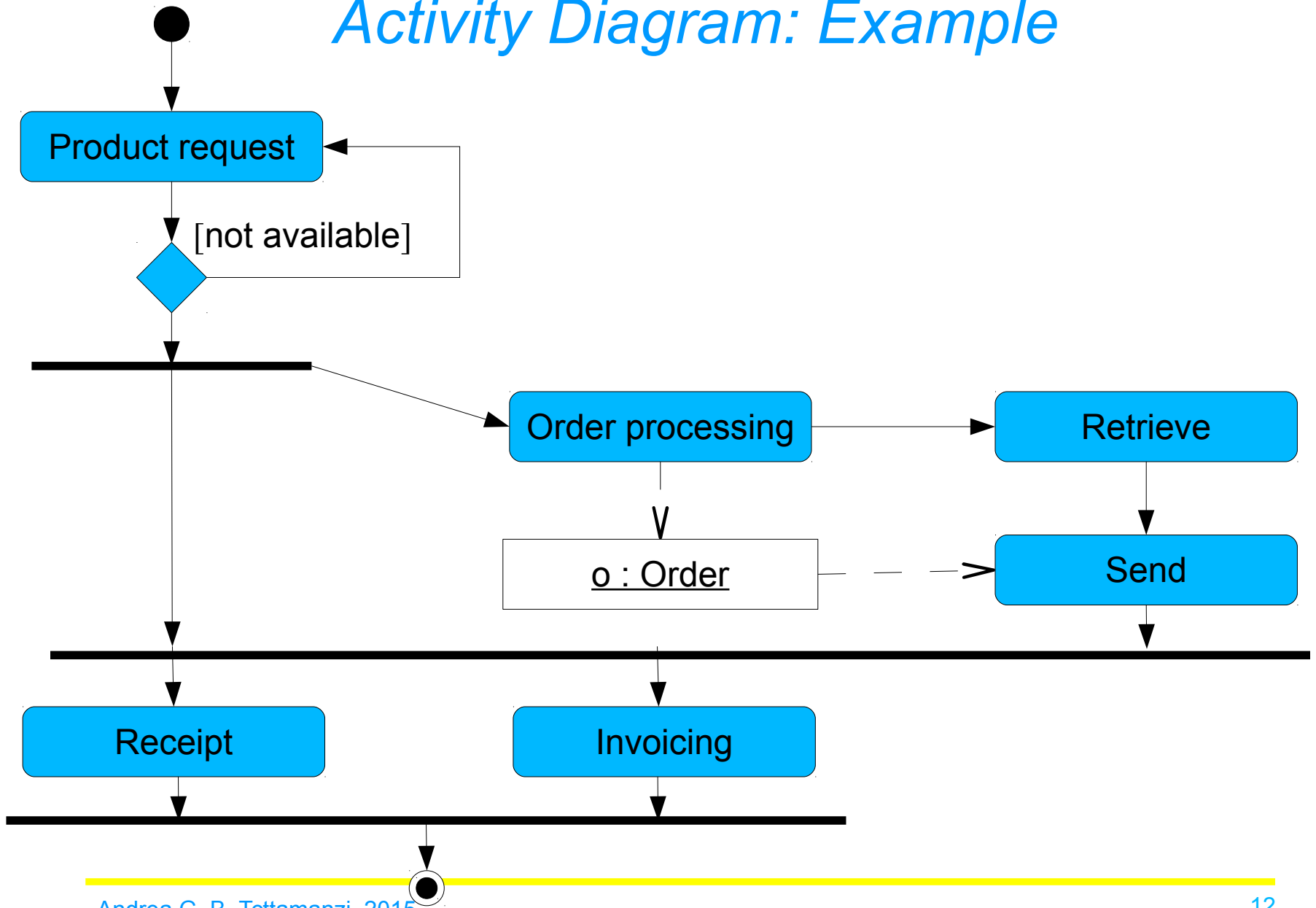


Object flow

Name : Class

Object

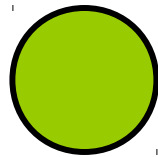
Activity Diagram: Example



Petri Nets

- Two interesting features
 - Formal definition, allowing formal verification
 - Intuitive graphical representation
- Two types of Petri Nets
 - Place-transition
 - State-event

Petri Nets: Graphical Elements



condition



event

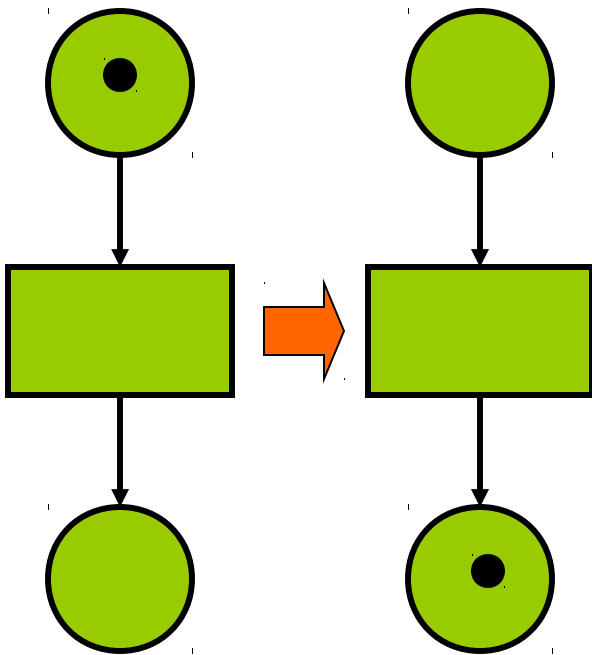


token

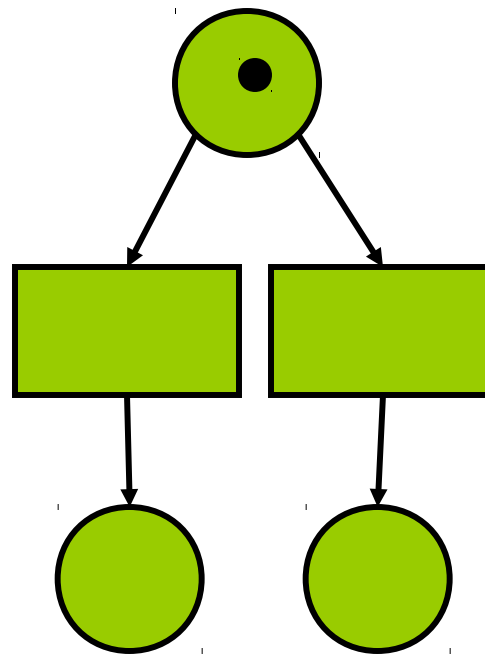


flow

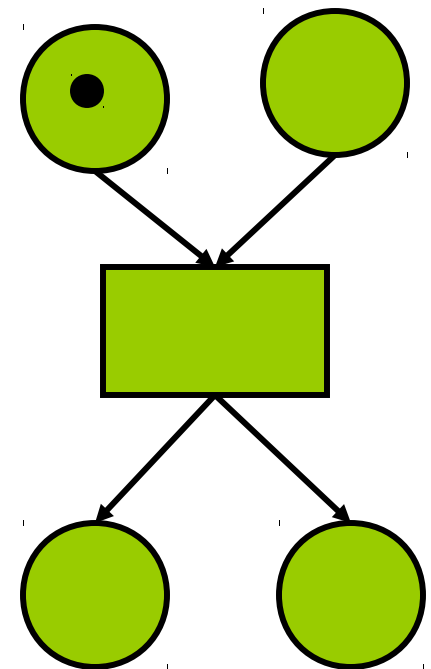
State-Event Petri Nets



Firing of a transition

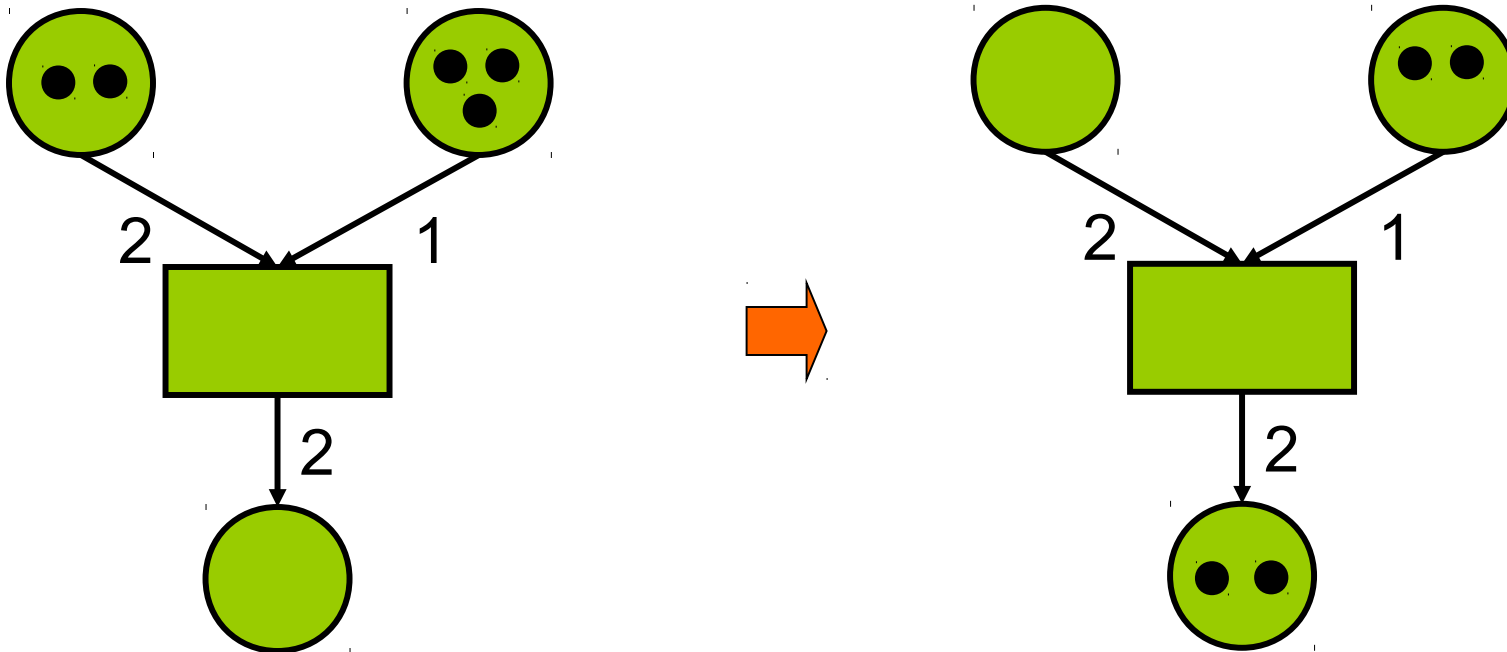


conflict



synchronization

Place-Transition Petri Nets

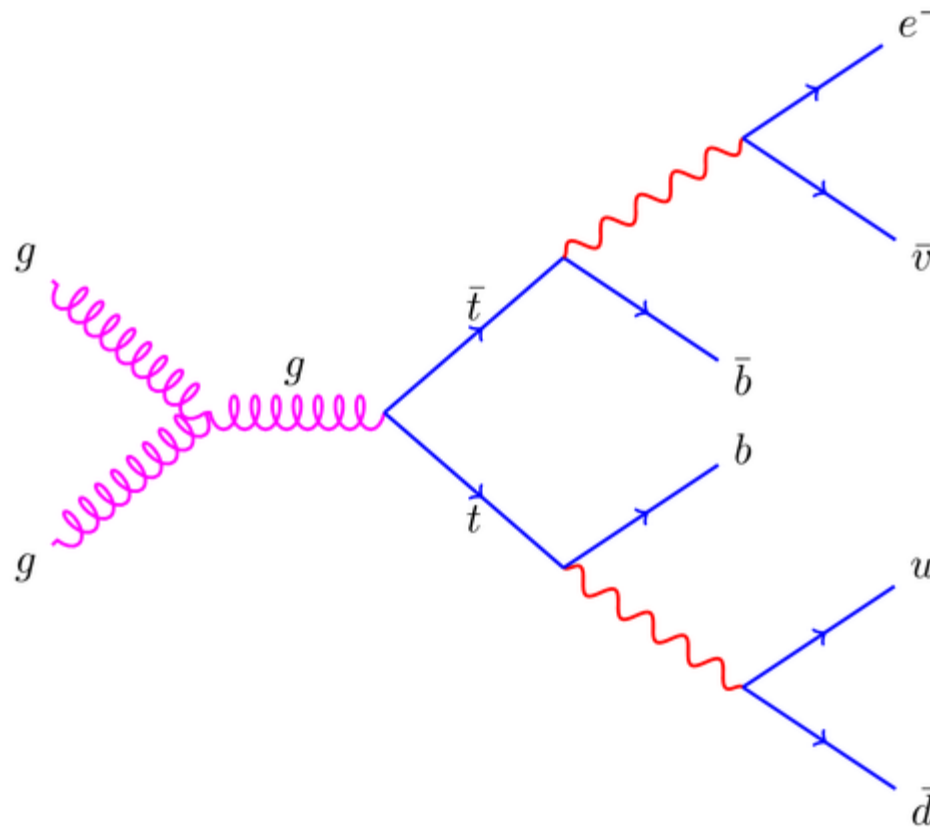


**Firing of a
transition**

Actor Event Diagrams

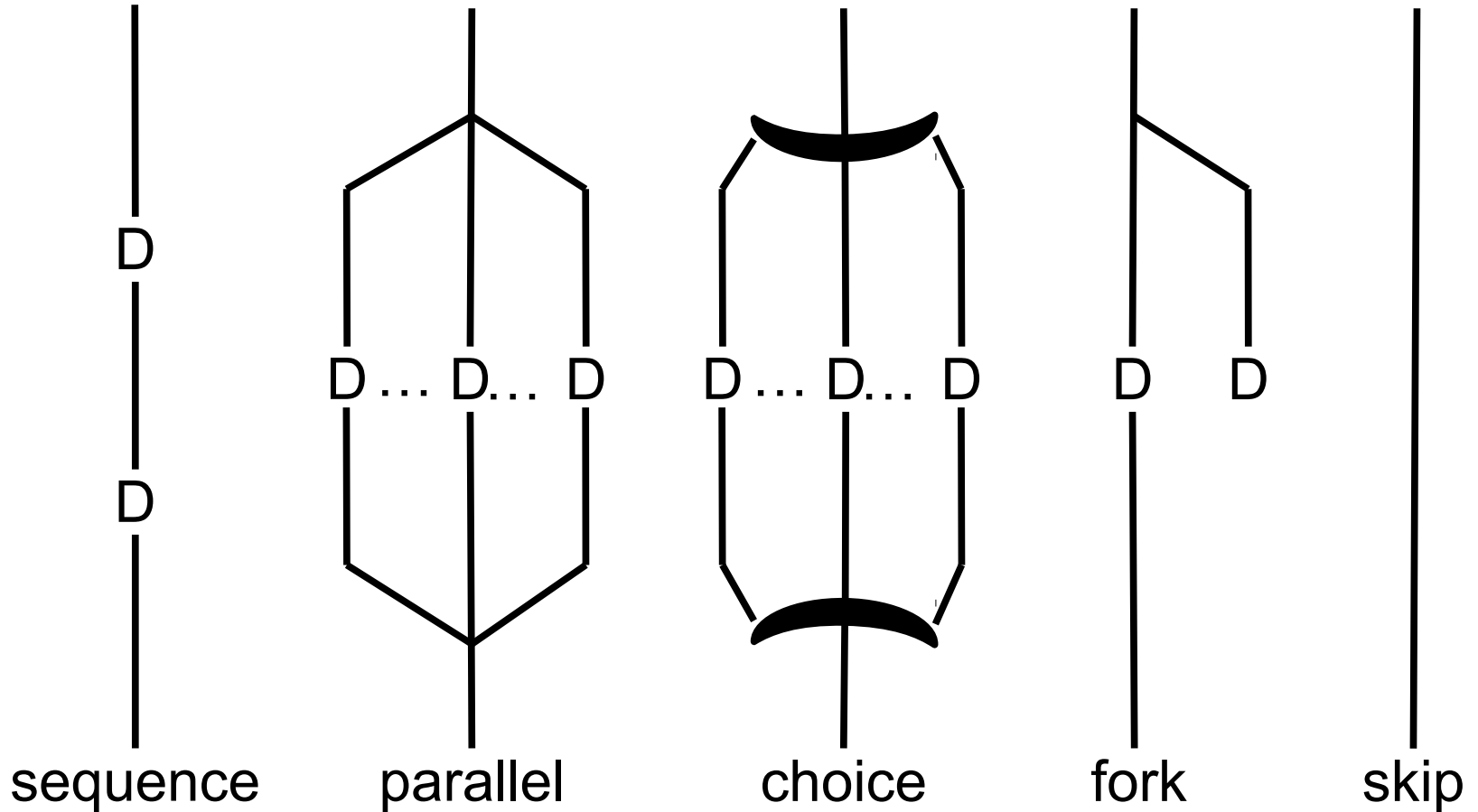
- Based on the Actor model
 - A mathematical model of concurrent computation
 - Actors are the universal primitives of parallel computation
 - Actors = Processes
 - Actors exchange messages
 - Inspired by Physics, including general relativity and quantum mechanics
 - Asynchronous communication
 - Control structures as message passing patterns
- Actor Event Diagrams remind Feynman Diagrams.

Feynman Diagrams



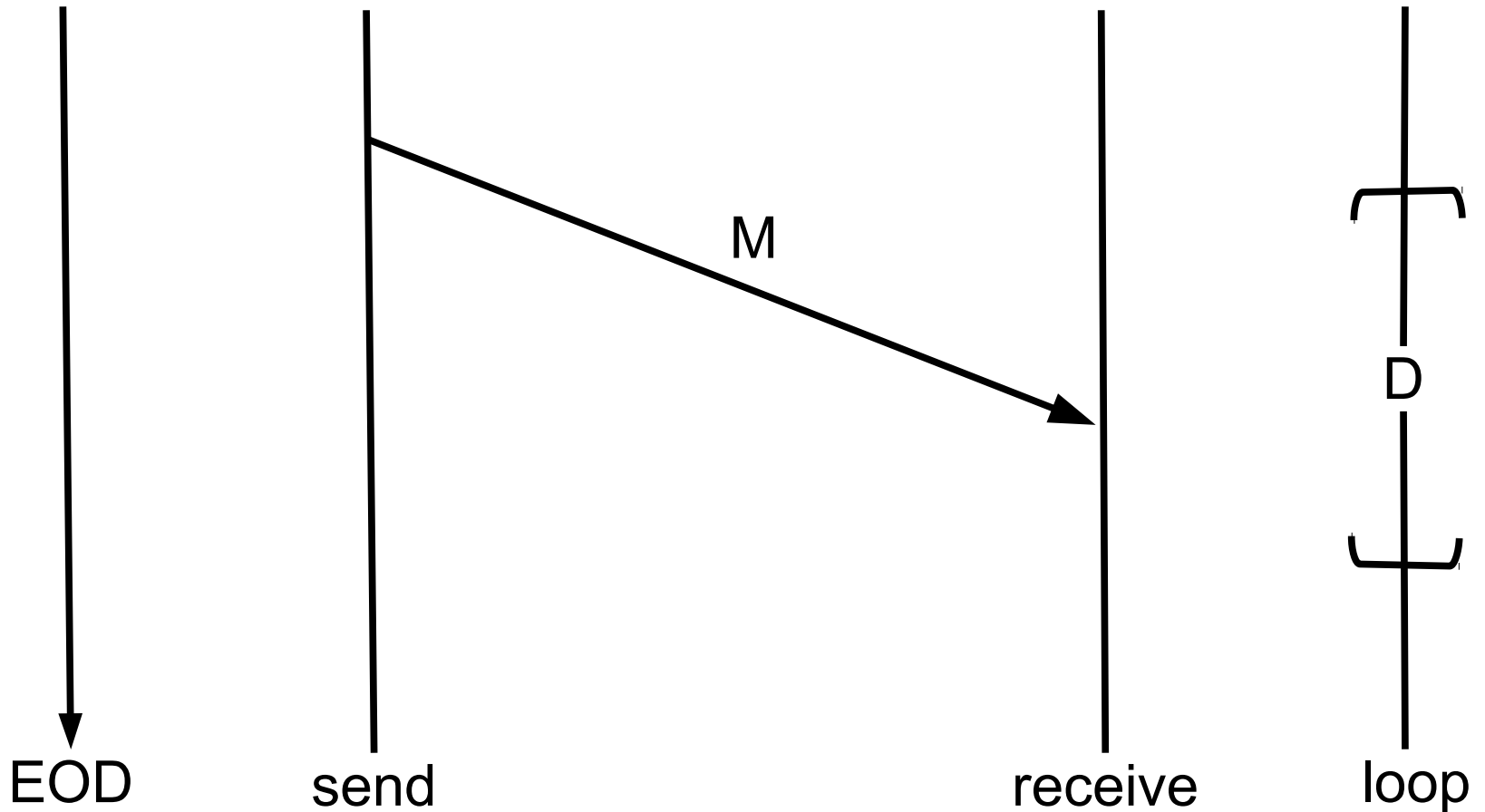
D = diagram

Actor Event Diagrams



D = diagram

Actor Event Diagrams



Algorithmic Skeletons

- Also known as Parallelism Patterns
- Take advantage of common programming patterns to provide an abstract description of parallel and distributed applications
- Some important skeleton patterns:
 - FARM, PIPE, FOR, MAP, D&C, WHILE, IF, SEQ
- Many libraries implement algorithmic skeletons

Algorithmic Skeletons

- FARM (or master-slave): task replication and execution in parallel
- PIPE: staged computation, different tasks performed in parallel on different stages of the pipe
- FOR: fixed iteration
- WHILE: conditional iteration
- IF: conditional branching
- MAP: split task into subtask, execute in parallel, and merge
- D&C (divide & conquer): task recursively subdivided
- SEQ: tasks are executed sequentially

Design of Parallel Algorithms

- Is there a principled way to design parallel algorithms?
- Ian Foster proposed a design process in 4 stages
 - Partitioning
 - Communication
 - Agglomeration
 - Mapping
- Partitioning and communication focus on concurrency and scalability
- Agglomeration and mapping focus on locality and performance

Partitioning

- Decompose the computation and the data into several small tasks
- Data
 - Domain/Data Decomposition
 - Partition data into smaller units that can be distributed
- Algorithm
 - Functional Decomposition
 - Partition the algorithm into tasks that can be performed in parallel/concurrently

Communication

- Focus on the flow of information and coordination among the tasks
- Four design dimensions/decisions:
 - Local vs Global
 - Structured vs Unstructured
 - Static vs Dynamic
 - Synchronous vs Asynchronous

Agglomeration and Mapping

- Agglomeration
 - Group small tasks into larger tasks to improve performance
- Mapping
 - Assign tasks to processors
 - Minimize communication cost

Thank you for your attention

