

Petri Nets — Part 1

Basic Concepts & System Execution

BSc Course in Electronic & Computer Engineering

What is a Discrete Event System (DES)?

Continuous vs Discrete

- Continuous Systems: Physical variables change smoothly all the time (like car speed or temperature).
- Discrete Event Systems: The state stays constant and changes instantly **ONLY** when a specific event happens.
- An Event is a sudden action (pushing a button, arriving packet, finishing a task).

Real-World Examples

- Coffee Vending Machine: Waits for a coin, then instantly prepares coffee.
- Automated Factory: Robot waits for a part, then moves it to the assembly line.
- Computer Network: Server waits for data packets, then processes them in order.

Why Do We Need Petri Nets?

Limits of Simple Diagrams

- Traditional state machines describe one single action at a time.
- In real factories or software, many actions happen AT THE SAME TIME (in parallel).
- Simple state diagrams become huge and confusing when systems grow.

The Petri Net Solution

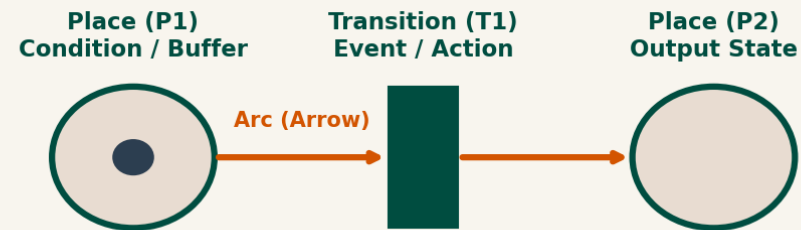
- Created by Carl Adam Petri in 1962.
- Provides a visual, easy-to-read map of parallel operations.
- Describes complex systems simply without state space explosion.

The 3 Main Building Blocks

Visual Components

- Circles = Places (P): Conditions, storage, or buffers.
- Rectangles = Transitions (T): Events, actions, or work steps.
- Arrows = Directed Arcs (F): Connections showing the flow.
- Black Dots = Tokens: Representing physical items, resources, or signals.

Petri Net Basic Building Blocks



Building Block 1: Places (Circles)

What is a Place?

- Places are drawn as CIRCLES.
- They describe passive conditions in the system.
- Examples of Places:
 - - 'Machine is Free'
 - - 'Buffer has 3 parts'
 - - 'Data packet in memory'

Key Rule for Places

- Places CANNOT connect directly to other Places!
- A Place must always connect to a Transition via an arrow.
- Places hold TOKENS to represent the current state.

Building Block 2: Transitions (Rectangles)

What is a Transition?

- Transitions are drawn as RECTANGLES or BARS.
- They represent active events or actions that change the state.
- Examples of Transitions:
 - - 'Start Processing'
 - - 'Press Start Button'
 - - 'Send Message'

Key Rule for Transitions

- Transitions CANNOT connect directly to other Transitions!
- A Transition must always be between Places.
- Transitions do NOT hold tokens — they only MOVE tokens!

Building Block 3: Arcs & The Bipartite Rule

The Bipartite Graph Rule

- Petri Nets are Bipartite Graphs: Arrows ONLY go from Place $\rightarrow$ Transition OR Transition $\rightarrow$ Place.
- Place $\rightarrow$ Place is FORBIDDEN. Transition $\rightarrow$ Transition is FORBIDDEN.

FORMULA & NOTATION: Tuple Notation

(P, T, F, M_0)

- P = Set of Places (Circles), T = Set of Transitions (Rectangles)
- F = Flow Arcs (Arrows connecting P to T , or T to P)
- M_0 = Initial Token Distribution (Starting State Vector)

Tokens & The Marking Vector M_0

What are Tokens?

- Tokens are black dots inside Places.
- They represent physical objects or active status:
 - - A raw part waiting in a box
 - - An available robot arm
 - - A user logged into a website

FORMULA & NOTATION: Marking Vector M

$$M = [M(P_1), M(P_2), \dots, M(P_n)]^T$$

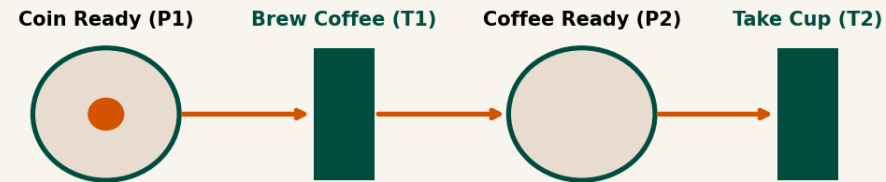
- The Marking Vector M lists the token count in every Place.
- Example: If Place 1 has 2 tokens and Place 2 has 0 tokens, $M = [2, 0]^T$.
- M_0 is the Initial Marking (the starting state of the net).

Real Example: Coffee Vending Machine

System States & Events

- Place P1: 'Coin Inserted' (Holds 1 Token).
- Transition T1: 'Brew Coffee' (Fires when coin is ready).
- Place P2: 'Coffee Ready in Tray'.
- Transition T2: 'Customer Takes Cup'.

Coffee Vending Machine Model



How Does a Petri Net Move? The Token Game

The Dynamic Evolution

- A Petri Net is NOT static — it moves and evolves over time!
- We play the 'Token Game' by firing enabled transitions.
- As events fire, tokens flow across places, changing the system state.

2 Golden Rules of Execution

- Rule 1 (Enabling): An event CAN happen only if all its input places have enough tokens.
- Rule 2 (Firing): When the event fires, it CONSUMES tokens from input places and CREATES tokens in output places.

Rule 1: When Can an Event Happen? (Enabling)

Enabling Conditions

- A Transition T is ENABLED (ready to fire) if EVERY place pointing to T contains at least 1 token.
- If even ONE input place is empty, the event CANNOT happen!
- An enabled transition is like a green traffic light.

Important Note

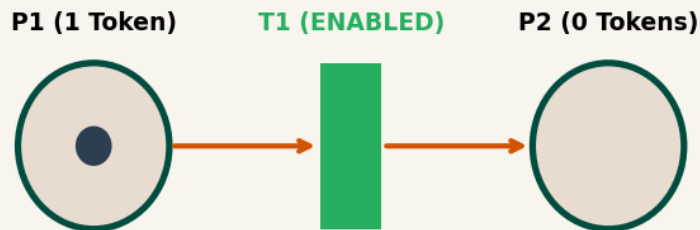
- Being ENABLED does not mean it fires immediately.
- It means the condition is satisfied and the event is READY to happen.

Rule 2: What Happens When an Event Fires?

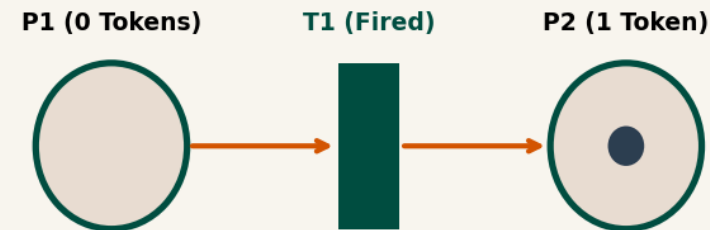
Token Consumption and Creation

- 1. Consumes (destroys) 1 token from each input place.
- 2. Creates (produces) 1 token in each output place.
- Firing is INSTANTANEOUS — it takes zero time in logical models.

BEFORE Firing (T1 Enabled)



AFTER Firing (Token Consumed & Created)



Formal Transition Function $\delta(M, t)$

FORMULA & NOTATION: State Update Equation

$$\delta(M, t) = M' \quad \text{where} \quad M'(p) = M(p) - W(p, t) + W(t, p)$$

- $\delta(M, t)$ is the Transition Function that calculates the new state M' from current state M .
- $M(p)$: Number of tokens in place p before firing.
- $W(p, t)$: Tokens consumed from input place p .
- $W(t, p)$: Tokens added to output place p .
- Tokens are NOT moved physical objects — they are destroyed at inputs and created at outputs!

Parallel Events: Concurrency Made Simple

What is Concurrency?

- When two or more transitions are enabled and do NOT share input places.
- They can fire independently in any order or simultaneously.
- No artificial sequential order is forced upon them.

Why it Matters

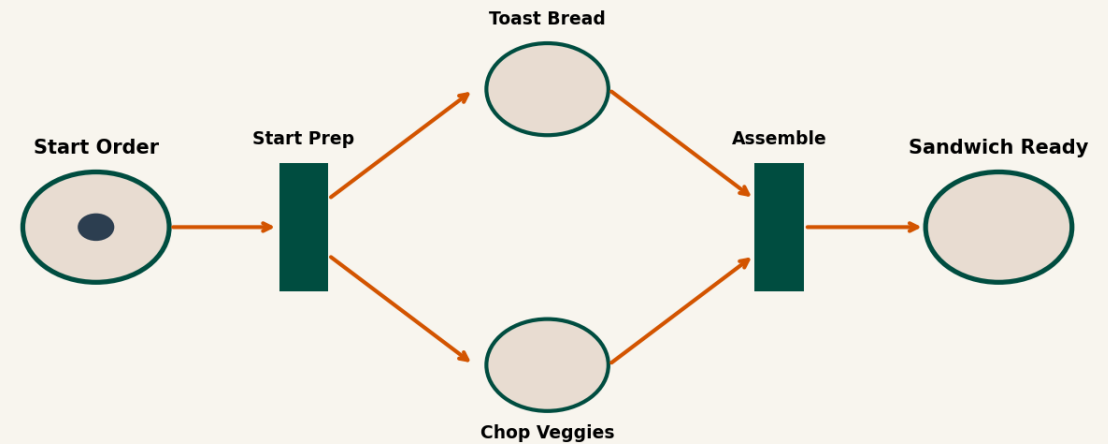
- Reflects real-world physical systems where multiple computers or robots work at the same time.
- Keeps models small and elegant.

Real Example: Making a Sandwich in Parallel

Parallel Kitchen Tasks

- 1. Start Prep splits order into two parallel branches.
- 2. Branch A: Toasting Bread.
- 3. Branch B: Chopping Veggies.
- 4. Assemble Sandwich waits for BOTH tasks to finish!

Parallel Tasks (Concurrency) in Petri Nets



Making Choices: Structural Conflict

What is a Conflict?

- When two transitions share the SAME input place with only 1 token.
- Both events are enabled, BUT firing one event CONSUMES the token and disables the other!
- Only ONE event can win the choice.

Non-Deterministic Choice

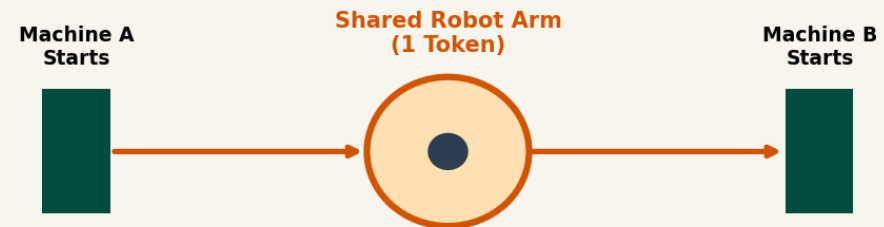
- The system chooses non-deterministically (randomly or depending on external environment).
- Models decisions, branching paths, or resource competition.

Real Example: Two Machines Sharing One Robot Arm

Resource Competition

- Place: Shared Robot Arm (Holds 1 Token).
- Machine A wants the robot.
- Machine B wants the robot.
- Whichever fires first gets the robot; the other must wait!

Resource Conflict / Competition



CONFLICT: Only ONE machine can take the robot token at a time!

Weighted Arcs: Needing More Than 1 Token

Numbers on Arrows $W(f)$

- By default, an arrow has weight = 1.
- If an arrow has weight = 3, the transition needs AT LEAST 3 tokens to enable.
- When it fires, it consumes exactly 3 tokens from the input place.

FORMULA & NOTATION: Arc Weight Rule

$$W(p, t) = k$$

- Input Condition: $M(p) \geq k$ tokens required to enable.
- Firing Action: $M'(p) = M(p) - k$ tokens after firing.
- Output Action: $M'(p_{out}) = M(p_{out}) + W(t, p_{out})$.

Real Example: Factory Batch Packaging

Accumulating Parts Before Packing

- Place P1: 'Produced Parts Buffer' (Accumulates single items).
- Arc Weight $W(P1, T1) = 4$: The packing machine needs a batch of 4 items.
- Transition T1 ('Pack Box') stays DISABLED when P1 has 1, 2, or 3 items.
- As soon as token count $M(P1) = 4$, Transition T1 becomes ENABLED and packs 1 box!

Limited Storage: Capacity Constraints $K(p)$

Finite Capacity Limits

- In physical systems, places cannot store infinite items.
- Place Capacity $K(p)$ defines maximum tokens allowed in place p .
- Transition T is enabled ONLY IF output place will not exceed $K(p)$.

FORMULA & NOTATION: Complementary Place Transformation

$$M(p) + M(p') = K(p)$$

- We can model capacity $K(p)$ using a Complementary Place p' .
- Initial tokens in $p' = K(p) - M_0(p)$.
- Tokens in p' represent EMPTY SPACES remaining in the buffer!

Simple Subclass 1: State Machines

Choice, No Concurrency

- Every transition has exactly 1 input arrow and 1 output arrow.
- Total token count in the net remains CONSTANT.
- Can model choices and decisions.
- CANNOT model parallel actions (concurrency).

FORMULA & NOTATION: State Machine Condition

$$|\bullet t| = 1 \text{ and } |t\bullet| = 1$$

- $|\bullet t|$: Number of input places to transition t .
- $|t\bullet|$: Number of output places from transition t .
- Equivalent to a standard Finite State Automaton!

Simple Subclass 2: Marked Graphs

Concurrency, No Choice

- Every place has exactly 1 input arrow and 1 output arrow.
- Can model parallel tasks and synchronization.
- CANNOT model choices or conflicts.

FORMULA & NOTATION: Marked Graph Condition

$$|\bullet p| = 1 \text{ and } |p\bullet| = 1$$

- $|\bullet p|$: Number of input transitions to place p .
- $|p\bullet|$: Number of output transitions from place p .
- Useful for pipeline processing and dataflow circuits!

Simple Subclass 3: Free-Choice Nets

Combining Choice and Concurrency Safely

- If two transitions share an input place, they must share ALL their input places.
- Separates choice from concurrency clearly.
- Prevents confusing interactions where choosing one path unexpectedly breaks another parallel path.
- Extremely popular in automated manufacturing and workflow design.

Execution Languages $L(M_0)$

Generating Words from Sequences

- By assigning action labels (like 'a', 'b', 'c') to transitions.
- Any valid firing sequence produces a word.
- The set of all valid words forms the Execution Language $L(M_0)$.

FORMULA & NOTATION: Language Power

$$L(M_0) = \{ w \in \Sigma^* \mid M_0 \rightarrow [w] \}$$

- Petri Nets can generate non-regular languages such as $L = \{a^n b^n \mid n \geq 1\}$.
- Finite Automata CANNOT recognize $a^n b^n$ because they cannot count.
- Petri Nets count easily by accumulating tokens!

Summary & Wrap-up of Part 1

Key Takeaways

- 1. Petri Nets are bipartite graphs with Places (Circles), Transitions (Rectangles), and Tokens.
- 2. System State is described by the Marking Vector M_0 .
- 3. Enabling Rule: Transition is ready if all input places have enough tokens.
- 4. Firing Rule: Consumes input tokens and creates output tokens instantly.
- 5. Models concurrency (parallelism) and conflict (choice) cleanly.
- Next in Part 2: Analyzing system behavior, boundedness, liveness, and deadlock avoidance!