Applications of Petri Nets

Presenter: Chung-Wei Lin

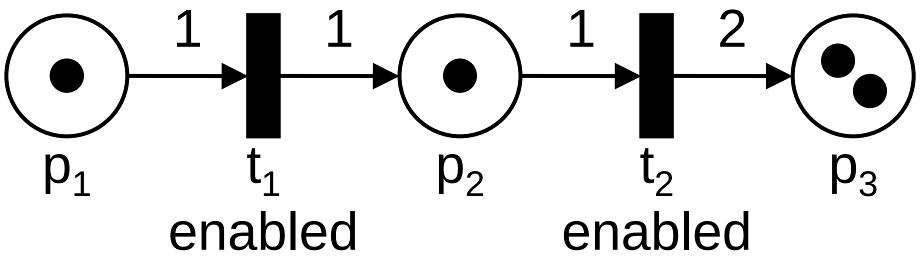
2010.10.28

Outline

- Revisiting Petri Nets
- Application 1: Software Syntheses
  - Theory and Algorithm
- Application 2: Biological Networks
  - Comprehensive Introduction
- Application 3: Supply Chains
  - Example and Experiment
- Summary

Definition of a Petri Net

- A 3-tuple (P,T,F)
  - P: set of **places**
  - T: set of **transitions**
  - F: $(P \times T) \cup (T \times P) \rightarrow N$, weighted flow relation



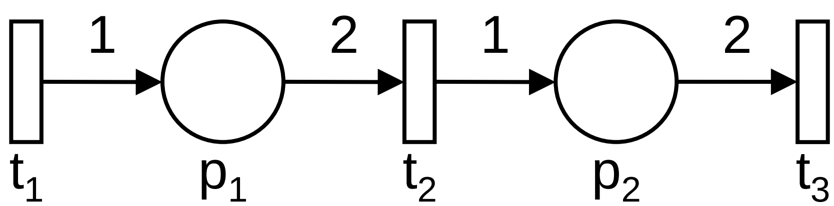
- How does it work?
  - Each place holds some (≥ 0) **tokens**
  - A transition is **enabled** if its input places contain at least the required # of tokens
  - The **firing** of an enabled transition results in
    - Consumption of the tokens of its input places
    - Production of the tokens of its output places

More about Petri Nets

- **Marking**
  - A vector representing the number of tokens in all places
- Properties of Petri Nets
  - Reachability (of a marking from another marking)
  - Boundedness
    - The numbers of tokens in all places are bounded
  - Conservation
    - The total number of tokens is constant
  - Deadlock-freedom
    - Always at least one transition can fire
  - Liveness
    - From any marking, any transition can fire sometime
  - Schedulability
    - The first paper will discuss this

T-Invariant and Finite Complete Cycle

- **T-invariant** is a vector s.t.
  - The i-th component is the number of firing times of transition t_i
  - The marking is unchanged if firing them so many times
  - However, it does not guarantee that a transition can be fired
    - Deadlock
- **Finite complete cycle** is a sequence of transitions s.t.
  - The marking is unchanged if firing the sequence



One T-invariant: (4,2,1)

Some finite complete schedules:

$<t_1,t_1,t_1,t_1,t_2,t_2,t_3>$

$<t_1,t_1,t_2,t_1,t_1,t_2,t_3>$

Outline

- Revisiting Petri Nets
- Application 1: Software Syntheses
  - Synthesis of Embedded Software Using Free Choice Petri Nets
- Application 2: Biological Networks
- Application 3: Supply Chains
- Summary

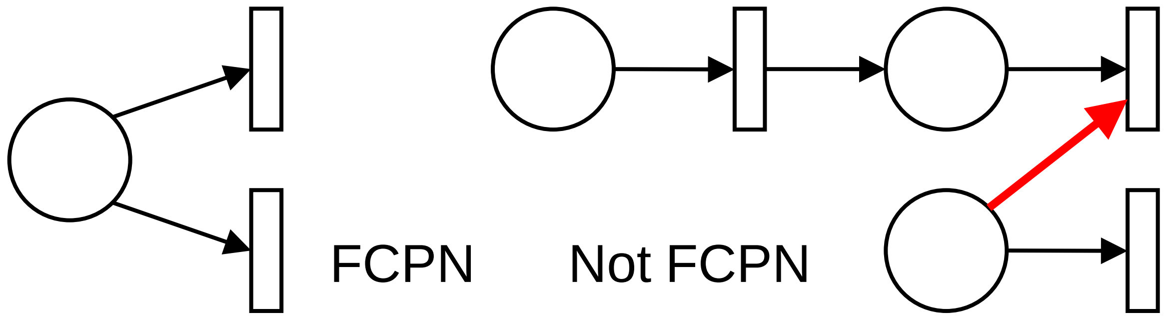
Static, Quasi-Static, and Dynamic Scheduling

- Scheduling problem
  - Mapping a functional implementation to real resources
  - Satisfying real-time constraints
  - Using resources as efficiently as possible
- Static scheduling
  - Specifications contain only data computations
  - The schedule can be completely computed at compile time
- Quasi-static scheduling
  - Specifications contain data-dependent controls, like if-then-else or while-do loops
  - The schedule leaves data-dependent decisions at run-time
- Dynamic scheduling
  - Specifications contain real-time controls

Free Choice Petri Net

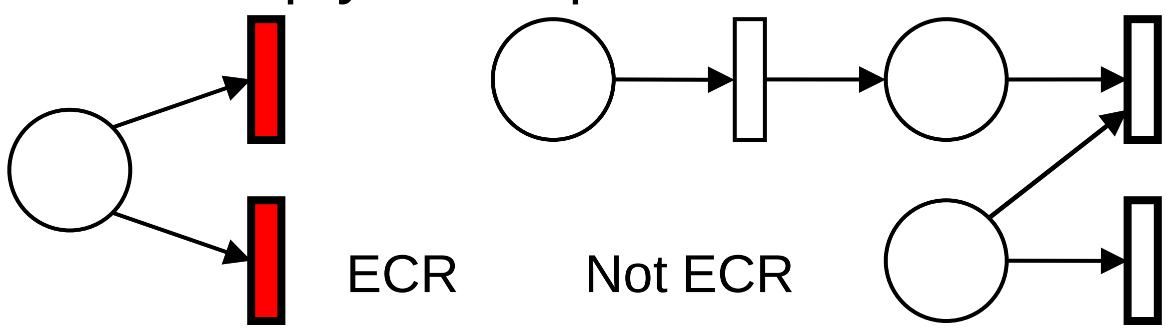
- Free Choice Petri Net (FCPN) is a Petri net such that every arc from a place is
  - A unique outgoing arc, or
  - A unique incoming arc to a transition

exactly an if-then-else structure!



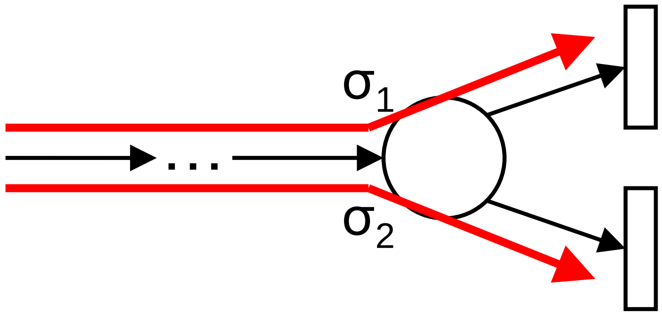
- Two transitions are in **equal conflict relation** (ECR) if their presets are non-empty and equal

branches of if-then-else structure



Valid Schedule (Set)

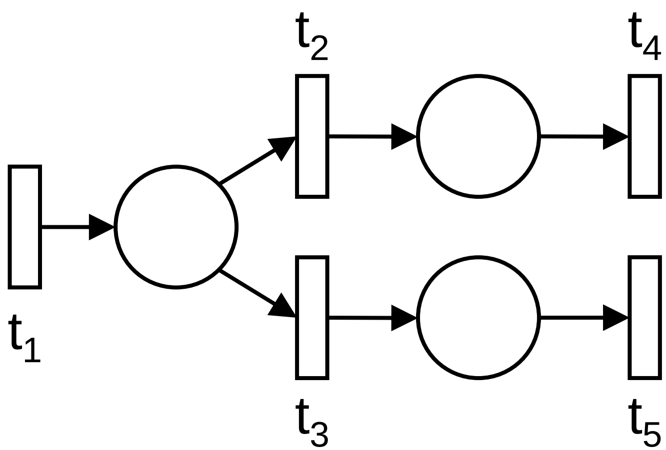
- Let
  - $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$ be a finite set
    - $\sigma_i = <\sigma_i^1, \sigma_i^2, \ldots>$ is a finite complete cycle containing all source transitions
- Σ is a **valid schedule (set)** if
  - For all (σ_i^j, t_k) s.t.
    - $\sigma_i^j \neq \sigma_i^h$ for all $h < j$
    - $\sigma_i^j \neq t_k$ and they are in equal conflict relation
  - Exist σ_l s.t.
    - $\sigma_l^m = \sigma_i^m$ for all $m \leq j$
    - $\sigma_l^m = t_k$ if $m = j$
- In words, a valid schedule is
  - A set of finite complete cycles for every possible outcome of a choice



σ_1 in Σ iff σ_2 in Σ

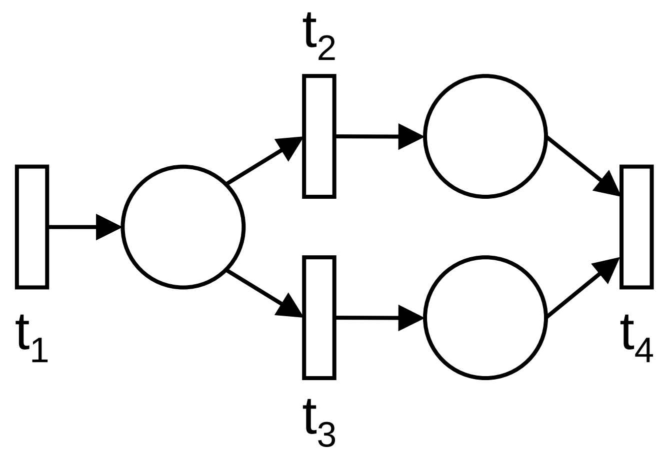
Quasi-Statically Schedulable

- Given
  - A FCPN N
  - An initial marking μ_0
- (N,μ_0) is **quasi-statically schedulable** if
  - There exists a valid schedule



$\Sigma = \{ <t_1,t_2,t_4>,<t_1,t_3,t_5> \}$

Schedulable



$\Sigma = \{ <t_1,t_1,t_2,t_3,t_4> \}$?

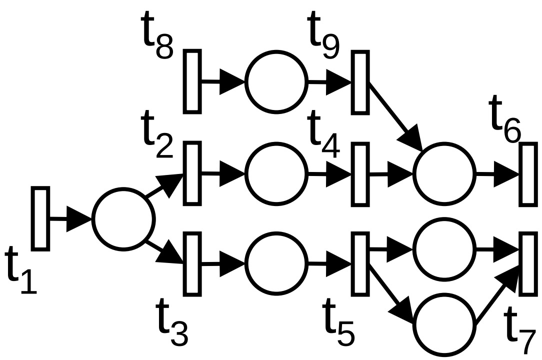
$\Sigma = \{ <t_1,t_1,t_2,t_3,t_4>,<t_1,t_1,t_3,t_2,t_4> \}$?

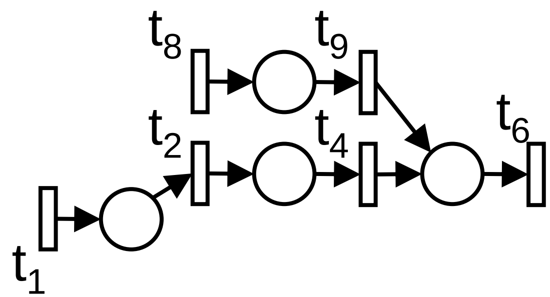
Non-schedulable

How to Find a Valid Schedule? Step 1

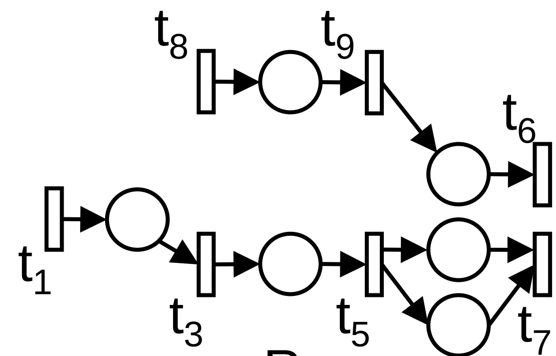
- **T-allocation** is a function that chooses exactly one transition for every place
- **T-reduction** associated with a T-allocation is a set of subnets generated from the image of the T-allocation

$A_1=\{t_1,t_2,t_4,t_5,t_6,t_7,t_8,t_9\}$ $A_2=\{t_1,t_3,t_4,t_5,t_6,t_7,t_8,t_9\}$



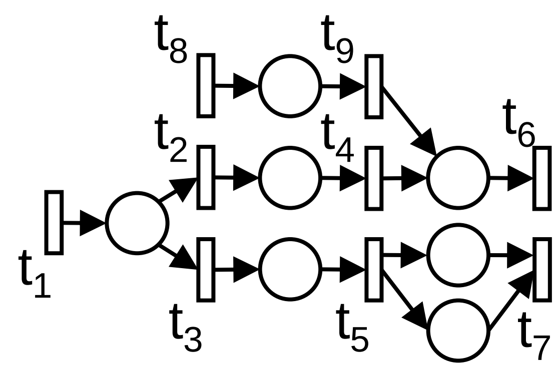


R_1



R_2

- Step 1: decompose a net into conflict-free components
  - Compute all T-reductions of the net
    - Reduction algorithm

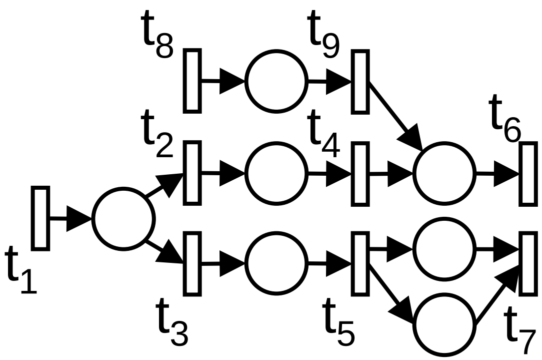


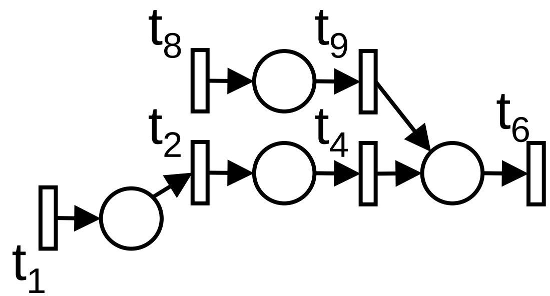
How to Find a Valid Schedule? Step 2

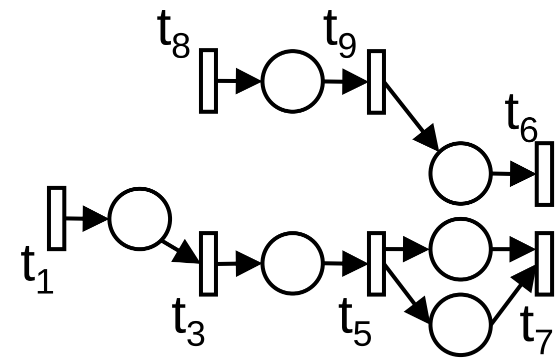
- A T-reduction is **schedulable** if
  - It has a finite complete cycle that
    - Contains at least one occurrence of every source transition of the net
  - (Definition 3.5)
- Step 2: check if every conflict-free component is statically schedulable
  - Apply the standard techniques for synchronous dataflow networks
    - Solve T-invariant equation
    - Check deadlock by simulation

How to Find a Valid Schedule? Step 3

- Given a FCPN, there exists a valid schedule if and only if every T-reduction is schedulable
  - Note that: a valid schedule → quasi-schedulable
- Step 3: derive a valid schedule, if there exists one
  - Compute the union of the finite complete cycles of all T-reduction



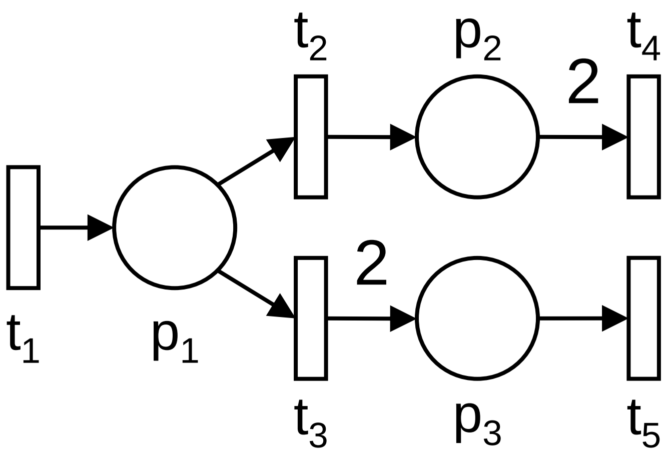




T-invariants (why needs two?)	(1,1,0,1,0,1,0,0,0) (0,0,0,0,0,1,0,1,1)	(1,0,1,0,1,0,1,0,0) (0,0,0,0,0,1,0,1,1)
finite complete cycle	$<t_1,t_2,t_4,t_6,t_8,t_9,t_6>$	$<t_1,t_3,t_5,t_7,t_8,t_9,t_6>$
valid schedule	$\{ <t_1,t_2,t_4,t_6,t_8,t_9,t_6>,<t_1,t_3,t_5,t_7,t_8,t_9,t_6> \}$	

Code Generation

- Derive an implementation directly from a valid schedule



valid schedule

$\{ \langle t_1,t_2,t_1,t_2,t_4 \rangle, \langle t_1,t_3,t_5,t_5 \rangle \}$

```
while ( true ) {
        t1;
        if ( p1 ) {
                t2;
                count ( p2 ) ++;
                if ( count ( p2 ) == 2 ) {
                        t4;
                        count ( p2 ) –= 2;
                }
        }
        else {
                t3;
                count ( p3 ) += 2;
                while ( count ( p3 ) >= 1 ) {
                        t5;
                        count ( p3 ) – –;
                }
        }
}
```

1→p_2→2 (annotates `if ( count ( p2 ) == 2 ) {`)

2→p_3→1 (annotates `while ( count ( p3 ) >= 1 ) {`)

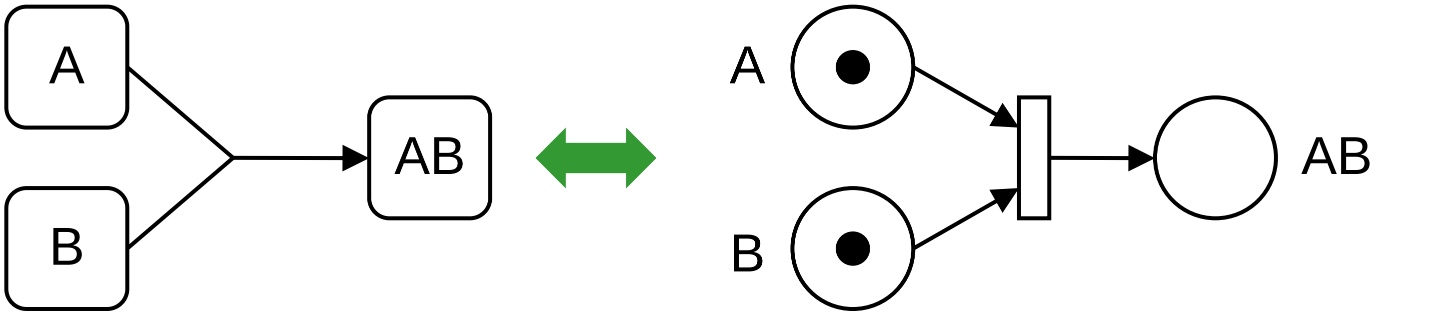
conflict transition (annotates `if ( p1 ) {` / `else {`)

Outline

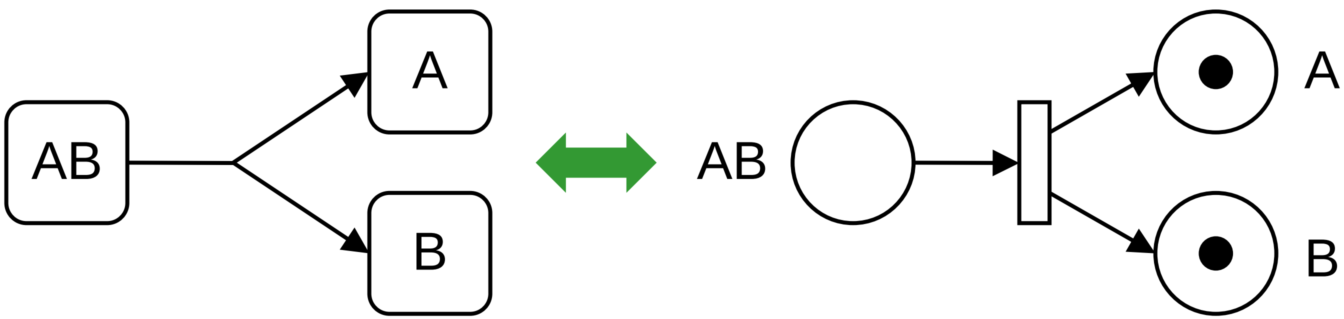
- Revisiting Petri Nets
- Application 1: Software Syntheses
- Application 2: Biological Networks
  - Petri Net Modeling of Biological Networks
- Application 3: Supply Chains
- Summary

Basic Modeling of Biological Reactions (1/2)

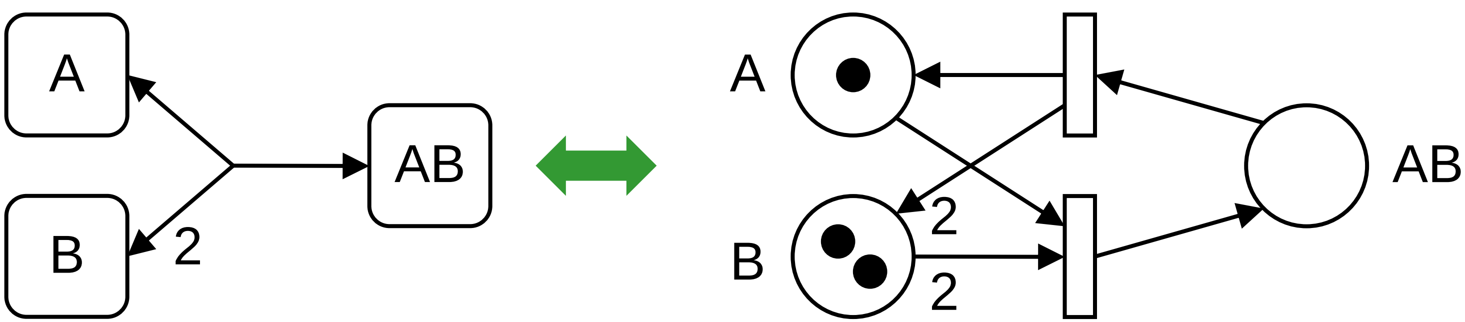
- Synthesis



- Decomposition

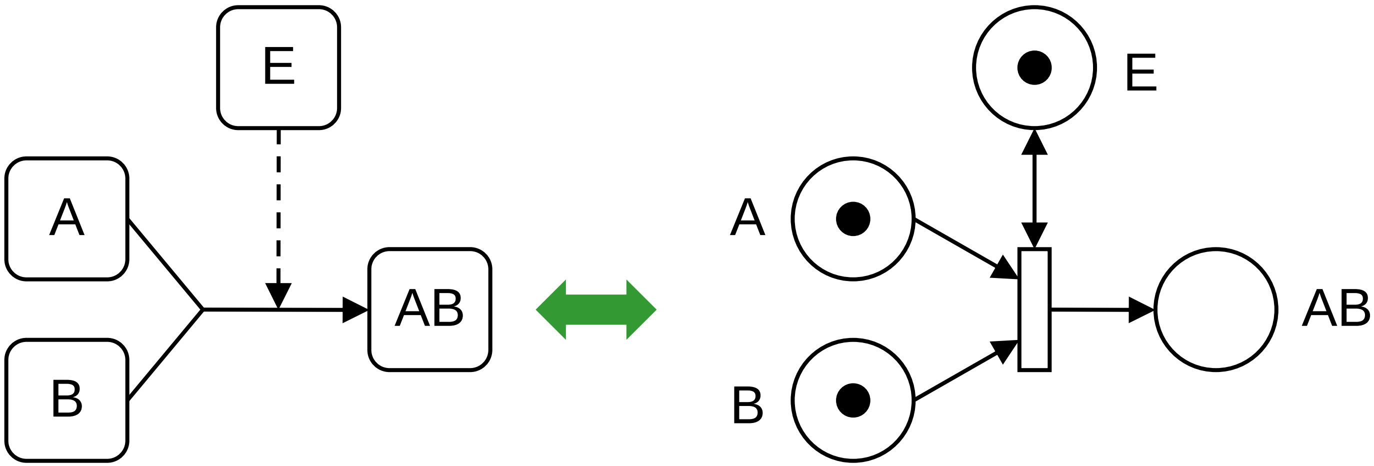


- Reversible reaction with stoichiometry

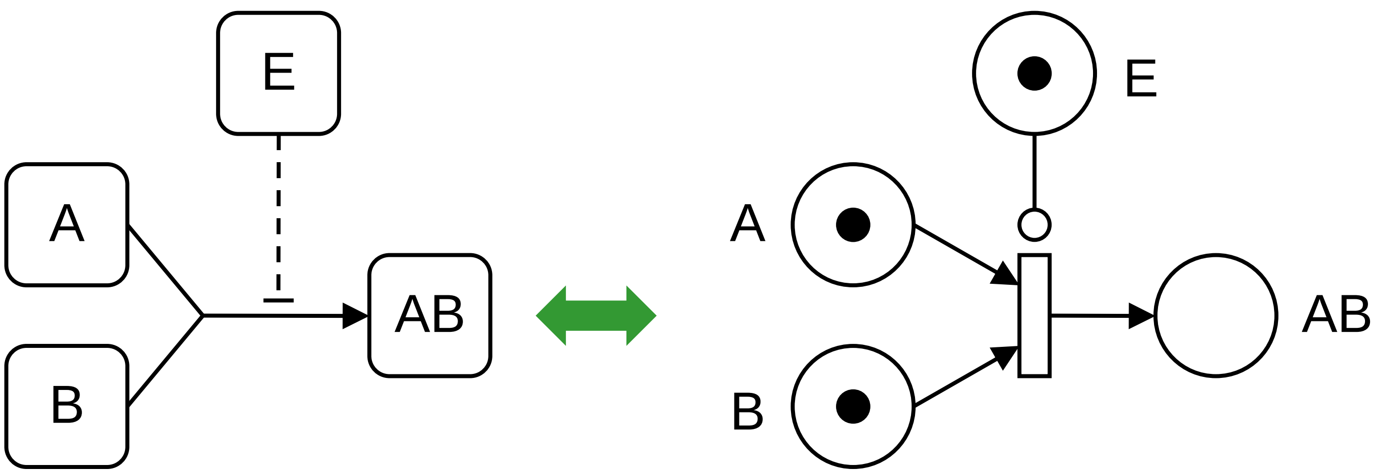


Basic Modeling of Biological Reactions (2/2)

- Catalyzed reaction



- Inhibited reaction



Extensions of Petri Nets

- Coloured Petri Net (CPN)
  - Assign data values to the token
  - Define constraints on the token values
- Stochastic Petri Net (SPN)
  - Transitions have exponentially distributed time delays
- Hybrid Petri Net (HPN)
  - Discrete and continuous places
    - Marked tokens and concentration levels
  - Discrete and continuous transitions
    - Determined and distributed delays
- Functional Petri Net (FPN)
  - Flow relations depend on the marking
- Hybrid Functional Petri Net (HFPN)

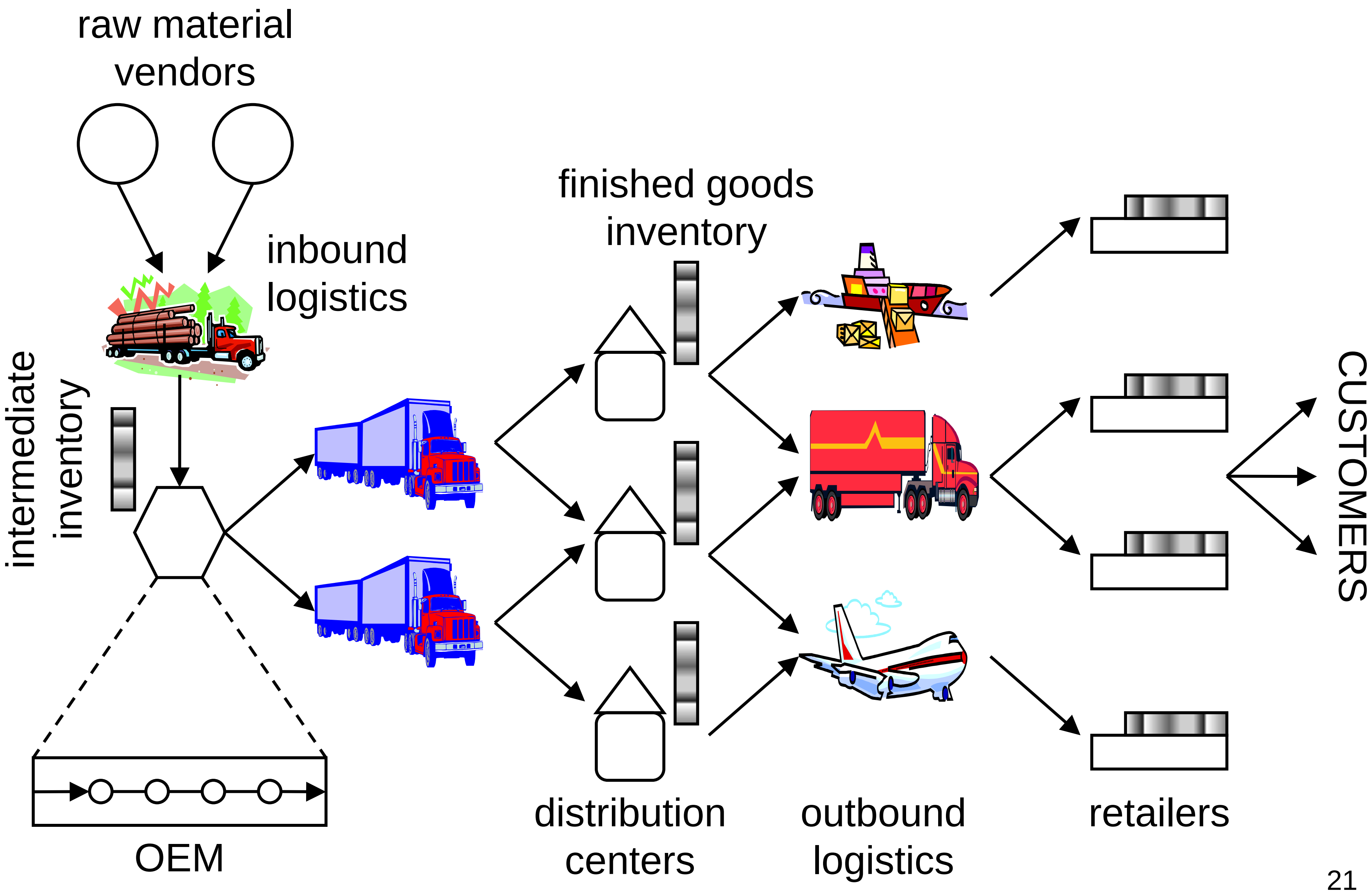
More Complicated Modeling

- Biochemical networks
  - Enzymatic reaction chain: CPN
  - Intrinsic noise due to low concentrations: SPN
  - More general pathway: FPN, HFPN
- Genetic networks
  - Response to genes rather than consumption and production
  - Switch Control: logical approach, CPN
  - Concentration dynamics: HPN, HFPN
- Signaling networks
  - Response to signal rather than consumption and production
  - Transition delay: timed PN, timed CPN, SPN
- Discussion
  - Tradeoff between expressiveness and analyzability
  - Spatial properties, hierarchical modeling, other modeling formalisms

Outline

- Revisiting Petri Nets
- Application 1: Software Syntheses
- Application 2: Biological Networks
- Application 3: Supply Chains
  - Performance Analysis and Design of Supply Chains: A Petri Net Approach
- Summary

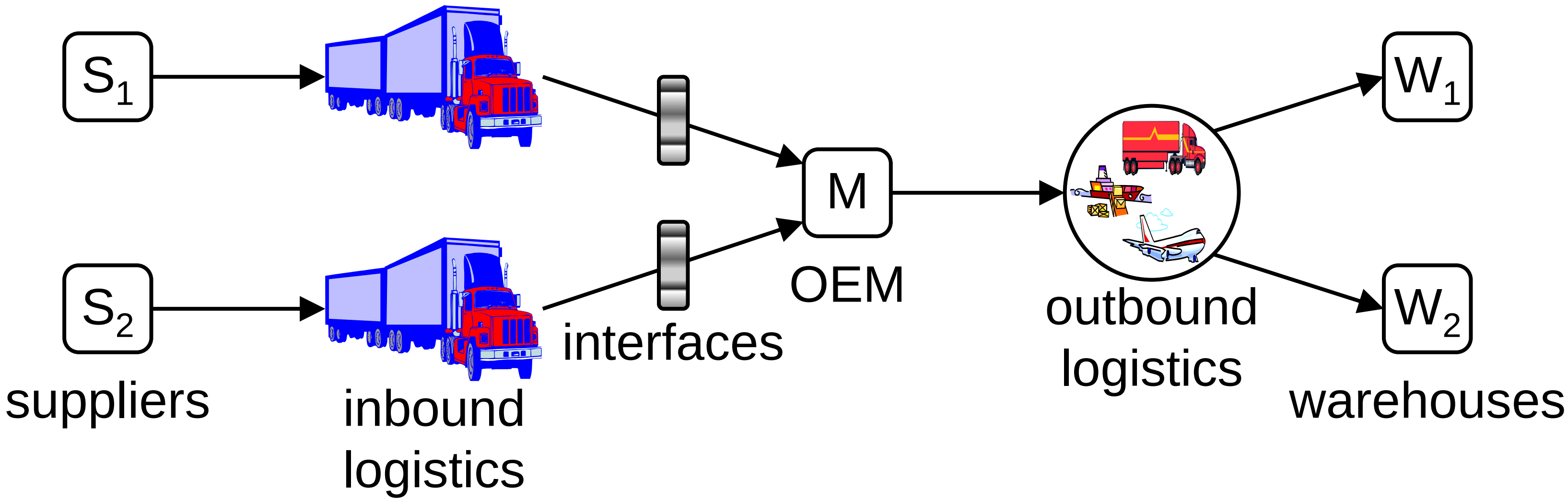
Supply Chain Networks (SCN)

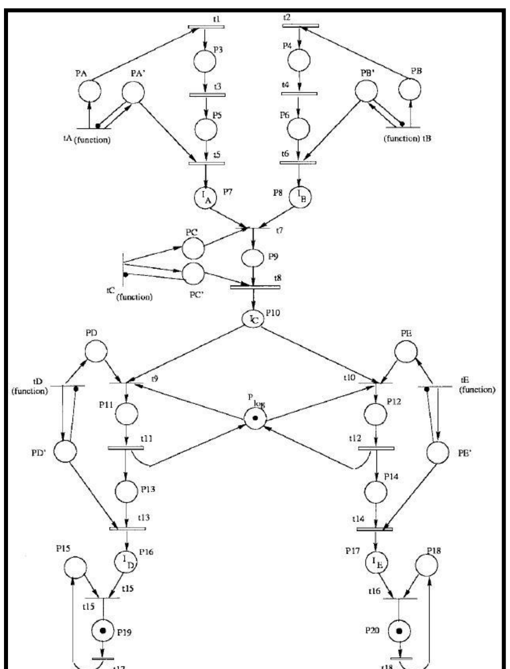


Configurations & Operational Models of SCN

- Configurations
  - Serial structure
  - Divergent structure: petroleum industry
  - Convergent structure: automobiles and air crafts
  - Network structure: computer industry
- Operational models
  - Make-to-stock (MTS)
    - Orders are satisfied from stocks of inventory of **finished goods which are kept at retail points**
  - Make-to-order (MTO)
    - **A confirmed order triggers the flow** of the supply chain
  - Assemble-to-order (ATO)
    - Before **decoupling point**, intermediate goods are made-to-stock
    - After decoupling point, goods are made-to-order
  - Tradeoff between holding cost and delayed delivery's cost

Performance Analysis – Modeling





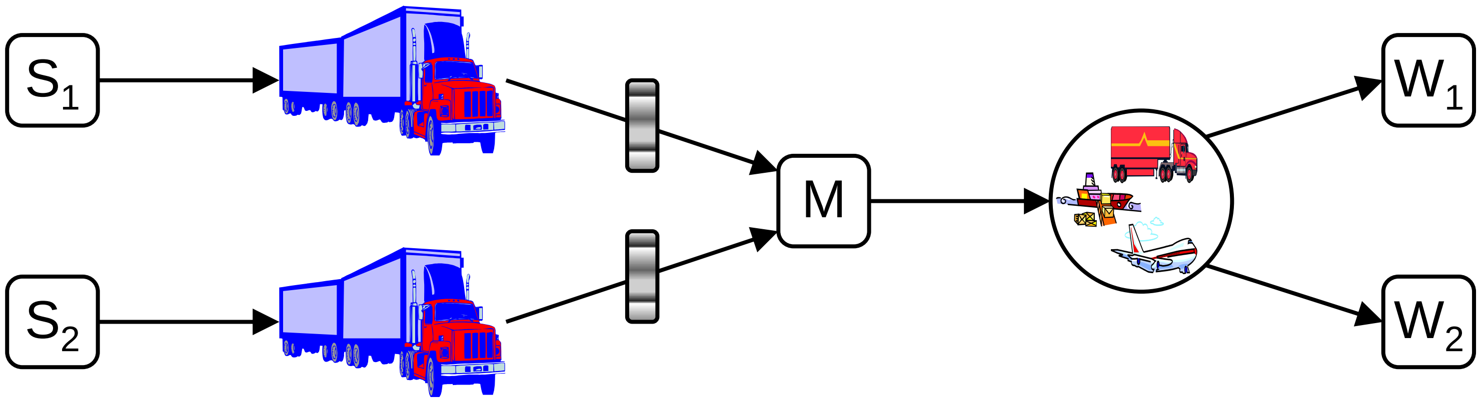
- Generalized Stochastic Petri Net (GSPN)
  - Random order request
  - Random logistics/interface time
- MTS, MTO, and ATO may have different structures & initial markings

Performance Analysis – Setting

- Cost function
  - Holding cost for inventories: H_I
  - Cost of delayed delivery: H_D
  - Vary the ratio of H_D and H_I from 1.5 to 40.0

- Apply Stochastic Petri Net Package (SPNP)
- Compare between make-to-stock (MTS) and assemble-to-order (ATO)

Performance Analysis – Experiment 1



- Change arrival rate of end products on (W_2)

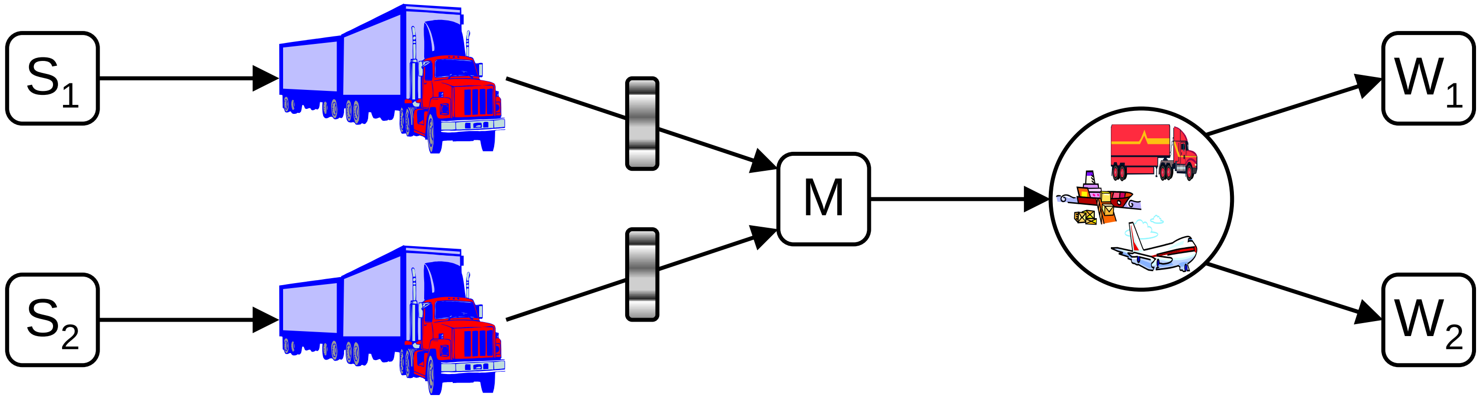
Arrival Rate (W_2)	Total Cost			
	H_D / H_I = 1.5 (Expensive Holding)		H_D / H_I = 40 (Expensive Delay)	
	MTS	ATO	MTS	ATO
0.8	22.421	19.815	26.001	257.437
1.0	21.237	18.610	25.818	237.559
1.2	20.012	17.714	25.961	224.228
1.4	18.774	17.016	26.339	214.675

MTS > ATO reasonable

U-shape why?

MTS < ATO reasonable

Performance Analysis – Experiment 2



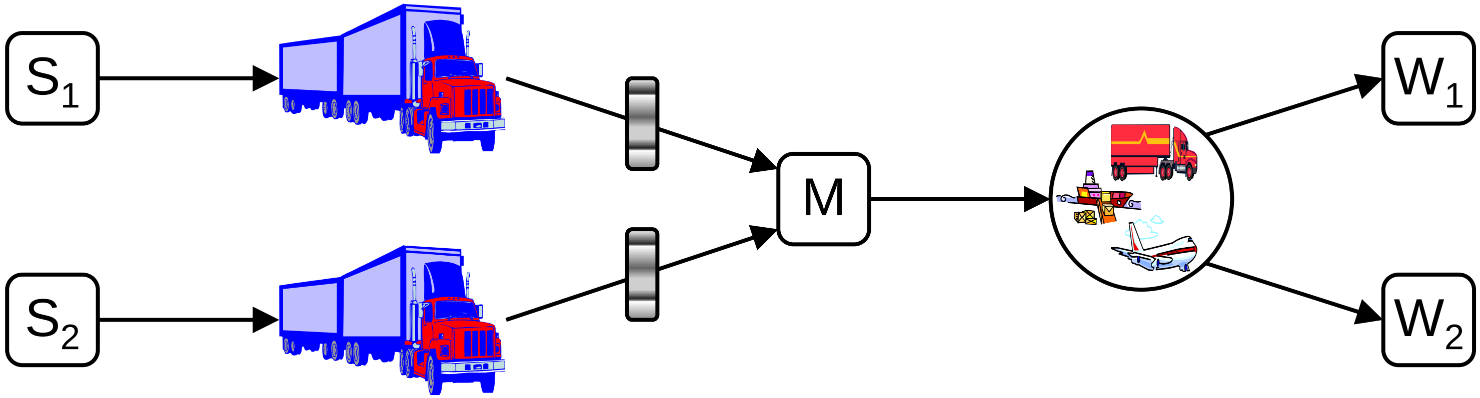
- Change targeted finished goods inventory (on M)

FGI (M)		Total Cost			
		MTS		ATO	
MTS	ATO	$H_D / H_I = 1.5$	$H_D / H_I = 40$	$H_D / H_I = 1.5$	$H_D / H_I = 40$
6	5	18.54	28.01	15.64	197.40
9	6	27.53	29.34	18.37	201.52
12	7	35.553	42.175	21.07	204.87
15	8	43.403	49.929	23.73	207.92

holding costs play more important roles

immune but useless

Performance Analysis – Experiment 3

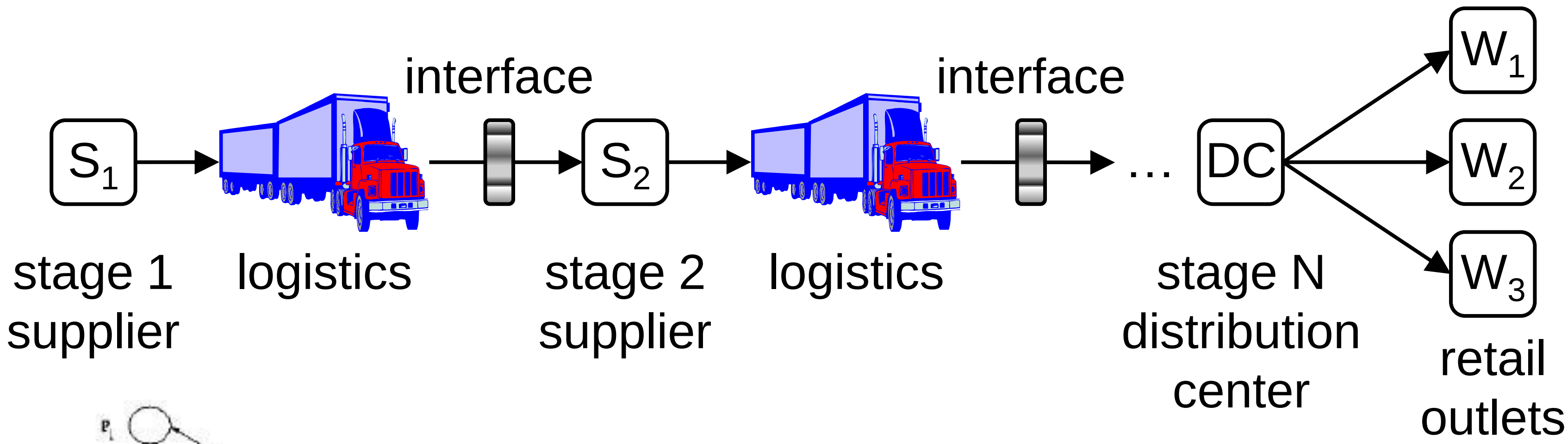


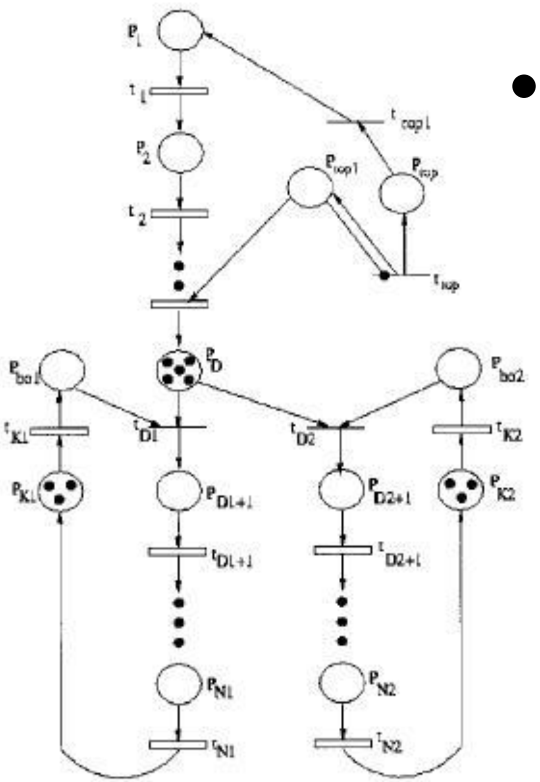
- Change interface times (from S_2 to M)

Interface Rate ($S_2 \rightarrow$ M)	Total Cost			
	$H_D / H_I = 1.5$ (Expensive Holding)		$H_D / H_I = 40$ (Expensive Delay)	
	MTS	ATO	MTS	ATO
4.0	22.566	15.542	24.934	197.185
5.0	22.651	15.640	24.981	197.360
6.0	22.709	15.705	25.038	197.502
8.0	22.780	15.785	25.109	197.659

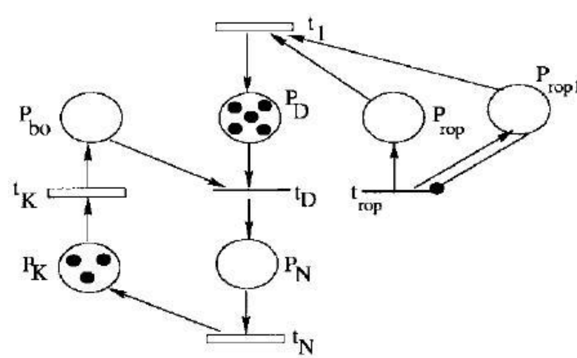
holding costs increase because interface S1→M is not fast enough to work with interface S2→M

Decoupling Point Location Problem – Modeling





- Integrated GSPN-queuing model
  - Amendable for integrated queuing network GSPN analysis and deriving aggregated facility by solving the original product from queuing network (PFQN)



Decoupling Point Location Problem – Setting

- Cost function
  - Holding cost (proportional to H_1)
    - H_1: holding cost for the first stage supplier
    - Increase as moving from the first stage supplier to the distribution center (H_1, $1.2H_1$, 1.2^2H_1, 1.2^3H_1, …)
  - Lead time cost (proportional to H_2)
    - H_2: average lead time cost per unit good per hour

- Consider 5 stage supply chain with the last stage being the retail outlet
- Set the decoupling point at stages 1, 2, 3, and 4
- Solve the PFQNs

Decoupling Point Location Problem – Experiment

Decoupling Point	Total Cost (Base Stock Policy)			
	Expensive Holding ←→ Expensive Delay			
	$H_2 / H_1 = 10$	$H_2 / H_1 = 30$	$H_2 / H_1 = 40$	$H_2 / H_1 = 50$
1	**3.596**	**5.940**	**8.285**	10.630
2	5.215	6.963	8.712	**10.461**
3	8.967	10.237	11.508	12.779
4	12.071	12.429	12.788	13.147

Decoupling Point	Total Cost (Reorder Point Policy)			
	Expensive Holding ←→ Expensive Delay			
	$H_2 / H_1 = 10$	$H_2 / H_1 = 30$	$H_2 / H_1 = 40$	$H_2 / H_1 = 50$
1	**3.427**	**5.853**	8.280	10.706
2	4.234	6.060	**7.886**	9.711
3	5.686	6.974	8.262	9.550
4	8.055	8.414	8.772	**9.131**

As delay cost increases, the decoupling point is moving to right

Outline

- Revisiting Petri Nets
- Application 1: Software Syntheses
- Application 2: Biological Networks
- Application 3: Supply Chains
- Summary

Summary

- Perti Net and its extension provide a wide range of applications
  - Software synthesis
  - Biological network
  - Supply chain