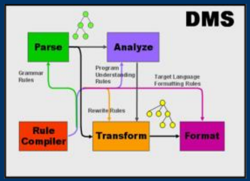
Automated Extraction of Business Rules and Models from Code

Ira Baxter, Ph.D., CEO/CTO
idbaxter@semanticdesigns.com

Friday, November 4, 2016 (11:30 am – 12:30 pm)
Room: Florentine III & IV

Example: Chemical Plants
Change: Migrate Process Control programs



- Problem: Trusted plant controller software running in end-of-life industrial control computers
    - Hundreds of different plants
    - Migrate to modern controllers
    - Recover abstract process control from “assembly code”

Some plants now converted

- Solution: (in progress)
    - Define abstractions in terms of dataflows with conditional implementations
    - DMS matches legacy code via dataflows (“Programmer’s Apprentice”)
    - Find consistent matching sets of abstractions
    - Reify abstractions to new controller code

3 Key Points from Today’s Presentation

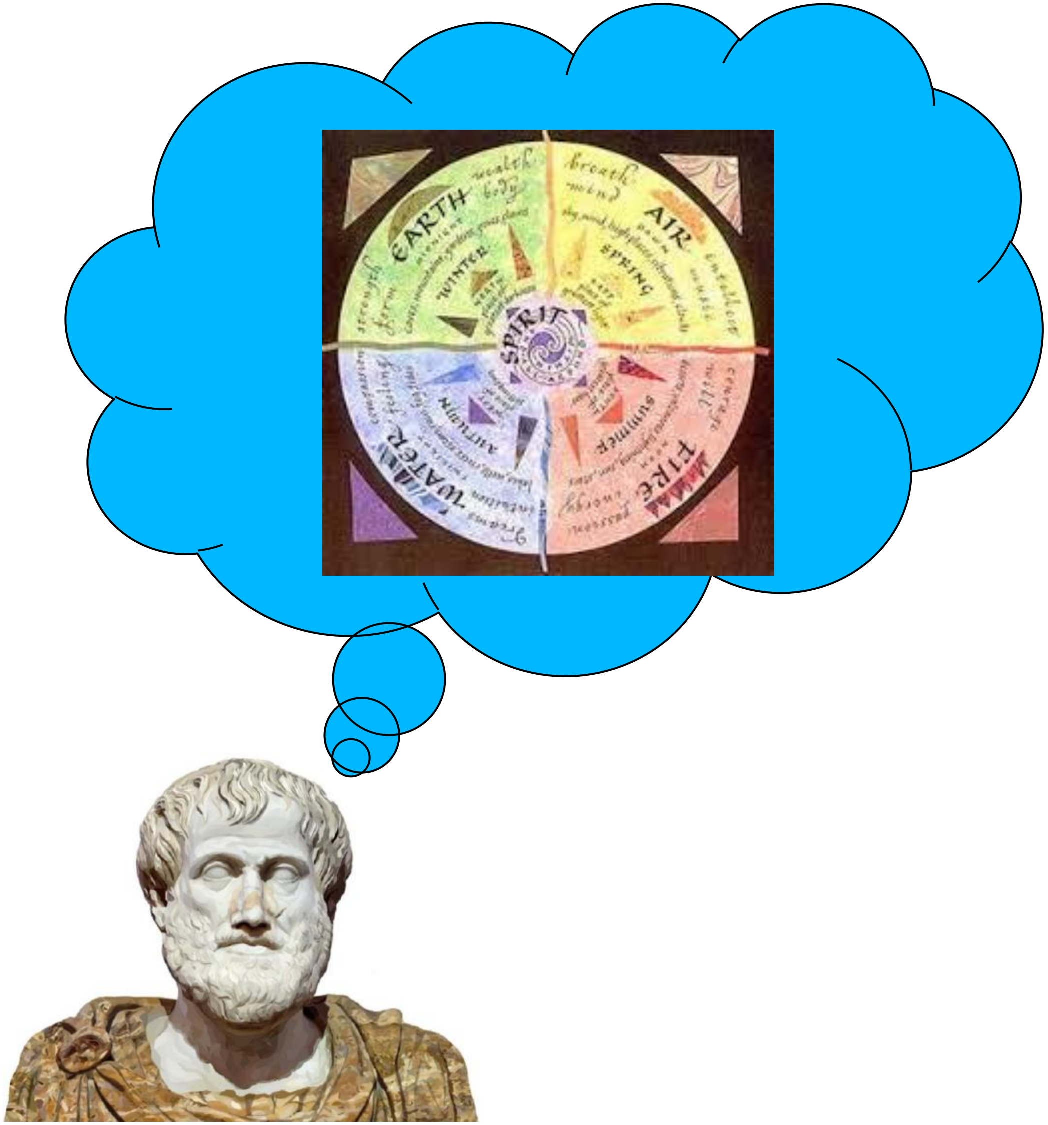
1. Define ***abstraction*** and how that is related to business rules
2. Show how computer code ***implements*** abstractions
3. Show how information flow ***and pattern matching can support finding abstractions*** in code enabling the extraction of business rules

Who am I?

Ira D. Baxter, Ph.D.

- **Research on**
  - **Software Reusability**
  - **Theory for formal program design capture and modification**
- **Founder/CEO/CTO of Semantic Designs (automated tools company)**
- **46 years building software tools**
  - **Operating Systems and Compilers for minis and micros**
  - **Program Analysis and Transformation Engines (DMS®)**
  - ***Usable* Parallel Programming languages (PARLANSE)**
- **Architect/Project lead on Automated Solutions to tough problems**
  - **Synthesis of parallel supercomputing codes for seismic simulation**
  - **Code generation of programs to run automobile factories**
  - **Translation of legacy mission software for B-2 Stealth Bomber**
  - **Automated architectural re-engineering of Boeing aircraft mission software**
  - **Impact analysis tools for large-scale mainframes: ANZ and US Social Security**
  - ***Model ("business rule") extraction for Dow Chemical factories***

The first model of the world might not be the right model



Aristotle

What can I do
if I believe this world model?

"… most substances are
compounds of these …"

Doh!

“A change in perspective is worth 80 IQ points” (Alan Kay, Xerox Parc)

Early Ideas on Elements

ert Boyle stated...

- A substance was an element unless it could be broken down to two or more simpler substances.

- Air therefore could not be an element because it could be broken down in to many pure substances.

Robert Boyle

What can I do
if I believe this world model?

Group → / ↓ Period	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1 H																	2 He
2	3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
3	11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
4	19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
5	37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
6	55 Cs	56 Ba		72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
7	87 Fr	88 Ra		104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Uut	114 Fl	115 Uup	116 Lv	117 Uus	118 Uuo

Lanthanides	57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
Actinides	89 Ac	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

Why do we care about "Business Rules" at all?

- **Our organizations are large, complex entities**
  - **We run them with complex processes largely composed of software**
  - **The software itself is made from accidental technologies *du juor / d'hier(!)***
  - **It is giant, complex, chaotic, and hard to understand**
  - ***… and we forgot what it all does***
- **How does management …**
  - **know what the organization is doing?**
  - **state what they want the organization to do?**
- **How can we change the software to do what management wants?**

- ***Answer: abstract what the "complex process" is doing***
  - **Hope the abstraction gets rid of accidental details**
  - **Hope the abstraction uses vocabulary management understands**

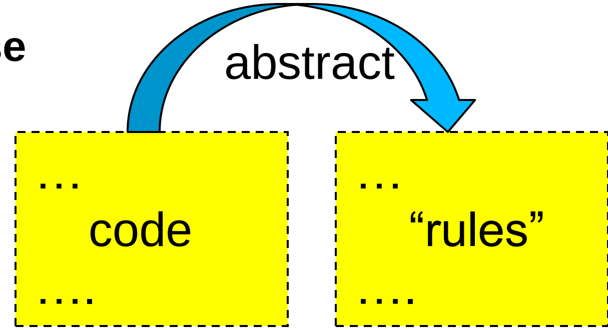
Abstraction? What on earth does that mean?

The essence of abstractions is *preserving information that is relevant* in a given context, and forgetting information that is irrelevant in that context.

– John V. Guttag (MIT Computer Science)

This means

- **teasing out what is important for a specific audience purpose**
- **from one system of vocabulary**
- **and translating it into another system of vocabulary**
- **easier to understand for that specific audience**



It also means:

leaving out the details that don’t matter.

Insight: Most of the way our software is implemented involves details that ultimately don’t matter;

all we want is the effect!

… isn’t it a bit weird that managers seem know this, and programmers don’t?

So… what is a “Business Rule”?

- **Requirements document (written in natural language [English])**
- **Z notation**
- **Semantics of Business Vocabulary and Business Rules (SBVR)**
- **Business Process Modeling Language**
- **Business Process Execution Language**
- **Decision Tables**
- **Decision Model**
- **Unified Modeling Language**
- **Flowcharts**
- **Domain Specific Languages**
- **Drools rules**
- **Random vendor result…?**

How can we agree on what a BR is if there are so many kinds?

So… what is a Business Rule?

- **Requirements document (written in natural language [English])**
- **Z notation**
- **Semantics of Business Vocabulary and Business Rules (SBVR)**
- **Business Process Modeling Language**
- **Business Process Execution Language**
- **Decision Tables**
- **Decision Model**
- **Unified Modeling Language**
- **Flowcharts**
- **Domain Specific Languages**
- **Drools rules**
- **Random vendor result…?**

How can we agree on what a BR is
if there are so many kinds?

Answer: They each focus on extracting certain information, abstracting away other information. Each is unique!

Problem for users: what specific information do you want?
Often chosen (poorly) for you by what a vendor happens to offer.

Key Background Concepts

(Business) State:

A set of facts true at a point in time about a (business) entity

State Model: ***The set of data describing a (business) entity***

- ***Have purchase_orders***
- ***Have warehouse_items***
- ***Have unpaid_invoices***
- ***Have available_cash***

Data Model: ***The structure and meaning of the data: Vocabulary***

- ***Structure == name and shape of the data***

  (PO is tuple <customer,itemID>)
- ***Meaning is the set of operations on the data and their results***

  Order(POs) → POs

  such that quantity (#) of POs increase

  Fulfill(POs,WIs,UIs) → <POs,WIs,UIs>

  such that #POs decreases, #WIs decreases, #UIs increase

  Payment(UIs,AC) → <UIs,AC>

  such that #UIs decreases, and amount AC increases

An algebraic (precise) specification (Spectrum) of business *data*

```
algebra Customer is String with
sorts:  customer ;
signatures:
  newCustomer(string) → customer ;
  name(customer) → string ;
axioms:
  name(newCustomer(string))==string ;
end
```

```
algebra ItemID is
sorts:  itemID ;
signatures: // discrete element algebra
axioms:
end
```

Shape == Schema

Meaning == Constraints On Operations

```
algebra PurchaseOrder is Customer+ItemID+Integer with
sorts: purchaseOrder ;
signatures:
  newOrder(Customer,ItemID,Natural) → purchaseOrder ;
axioms:
  orderingCustomer(newOrder(customer,itemID,integer)) == customer ;
  orderedItemID(newOrder(customer,itemID,integer)) == itemID ;
  orderedQuantity(newOrder(customer,itemID,integer)) == integer ;
end
```

```
algebra WarehouseItem is ItemID+Integer with
sorts: warehouseItem ;
signatures:
  quantityOnHand(itemID,natural) → warehouseItem ;
axioms:
  warehouseItemID(quantityOnHand(itemID,integer)) == itemID ;
  quantityavailable(quantityOnHand(itemID,integer)) == integer ;
end
```

An algebraic specification (Spectrum) of *business state*

```
algebra BusinessState is Set<PurchaseOrder>, Set<WareHouseItems>,
                   Set<UnpaidInvoice>, AvailableCash As Natural with
types: businessState

signatures:
  newCustomer(string) ➔ suffix(string)==“Inc.” ;
  newOrder(customer,itemID,desired) ➔ desired>0


  currentState(Set<PurchaseOrder>, Set<WareHouseItems>,
               Set<UnpaidInvoice>, AvailableCash) → businessState );
  purchaseOrders(businessState) → Set<PurchaseOrder> ;
...

  businessStartUp() → businessState ;
...
axioms:

   purchaseOrders(currentState(POs,WIs,UIs,AC) = POs ;

   businessStartUp() == currentState(empty,empty,empty,1000);

   currentState(newOrder(customer,itemID,quantity+Set<PurchaseOrder>,
                Set<WareHouseItems>,
                invoice(customer,anyprice)+Set<UnpaidInvoice>, cashonhand) ➔ false ;
         -- no new orders allowed until previous invoices are paid
```

Shape ==
Schema

Meaning ==
Constraints
On Operations

An algebraic specification (Spectrum) of *business actions (vocabulary)*

```
algebra Company is BusinessState with

signatures:
   order(businessState,customer,itemID,natural) ;
   fulfill(businessState) → businessState ;
   collect(businessState,customer) → businessState ;
   restock(businessSate,itemID,natural) → businessState ;
...

axioms:

   fulfill(currentState(newOrder(customer,itemID,desired)+Set<PurchaseOrder>,
                        quantityOnHand(itemID,available)+Set<WareHouseItems>,
                        Set<UnpaidInvoices>,cashonhand) ==
           currentState(Set<PurchaseOrder>,
                        quantityOnHand(itemID,available-desired)+Set<WareHouseItems>,
                        invoice(customer,desired*price)+Set<UnpaidInvoices>,
                        cashonhand))
      if available>=desired ;

   collect(currentState(Set<PurchaseOrder>,
                               Set<WareHouseItems>,
                               invoice(customer,amount)+Set<UnpaidInvoices>,cashonhand),
                  customer) ==
           currentState(Set<PurchaseOrder>,
                        Set<WareHouseItems>,
                        Set<UnpaidInvoices>,
                        cashonhand+amount)) ;
...
```

Meaning == Constraints On Operations

Two Fundamental Flavors of Rules

Constraints: sets of conditions over (business state) which must always be true

Already have some these in basic vocabulary

- ***Data Model:***
  The structure and meaning of the data (in a state) elements, and any constraints on those data elements

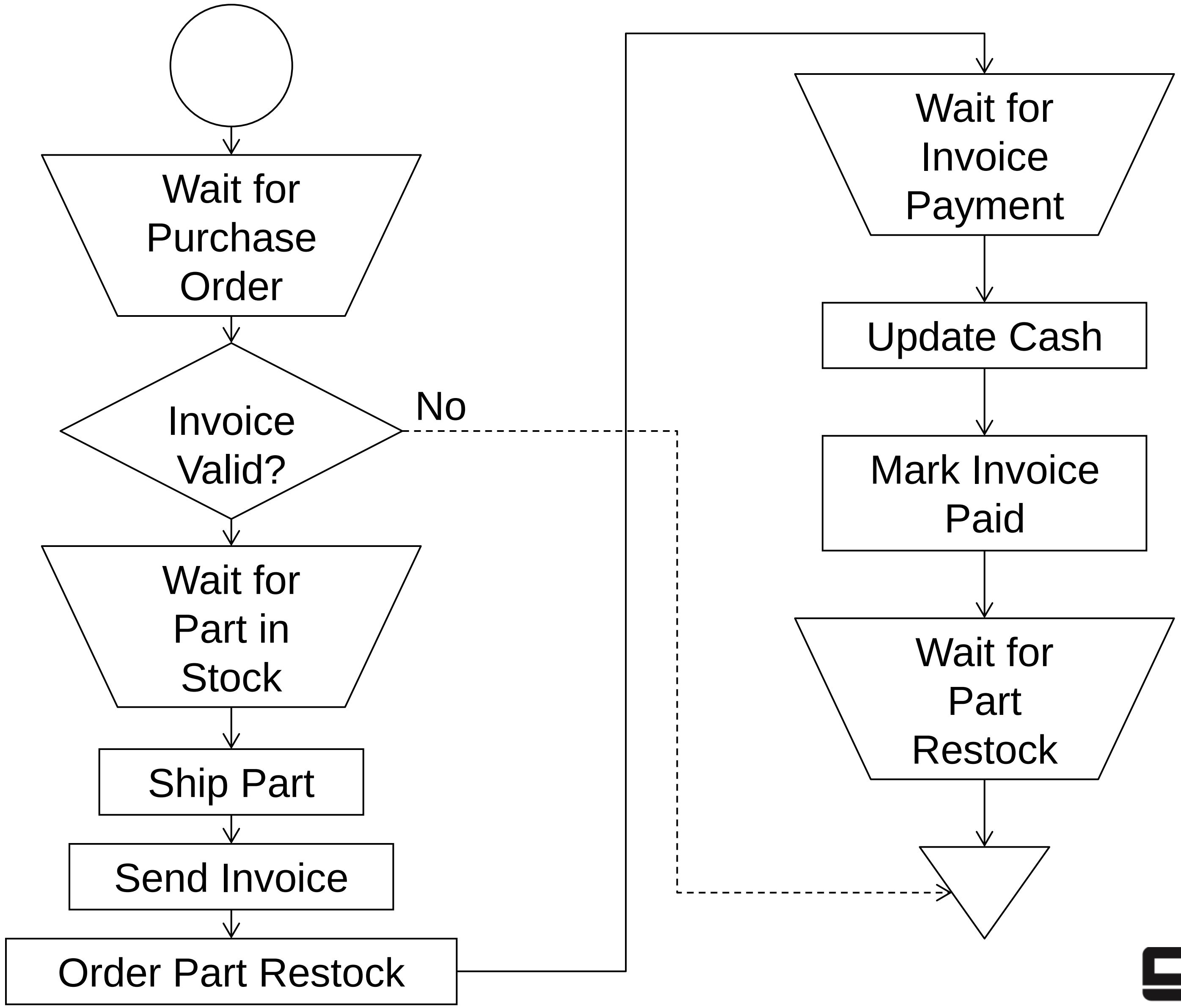
- ***State Model:***
  The set of data describing an (business) entity including constraints on the state of the business

Can state business actions in term of pre- and post- action constraints
Essence of "nonprocedural"

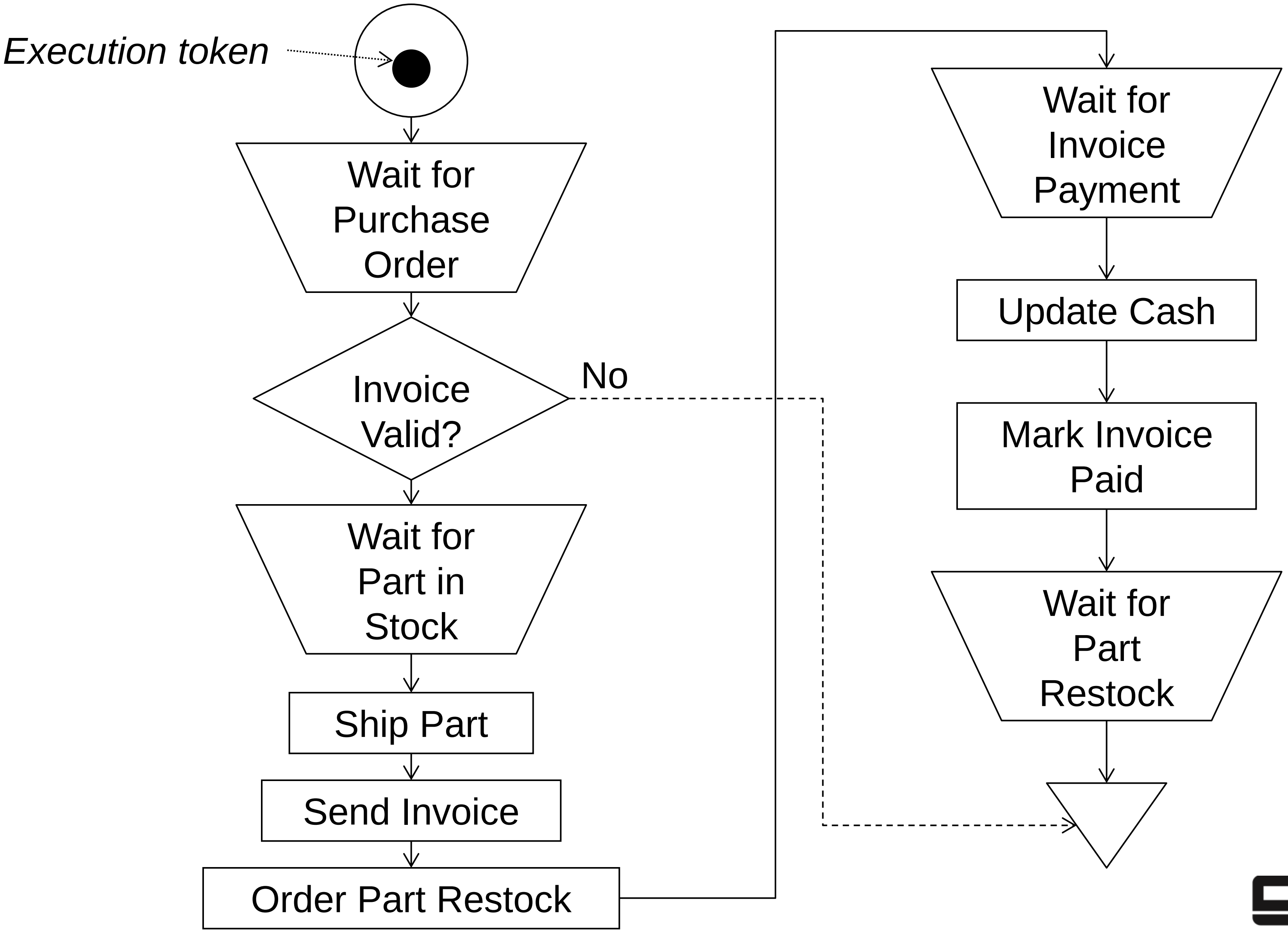
Procedures: reactions to new events to change business state in desired way

To be useful, these reactions must honor business constraints

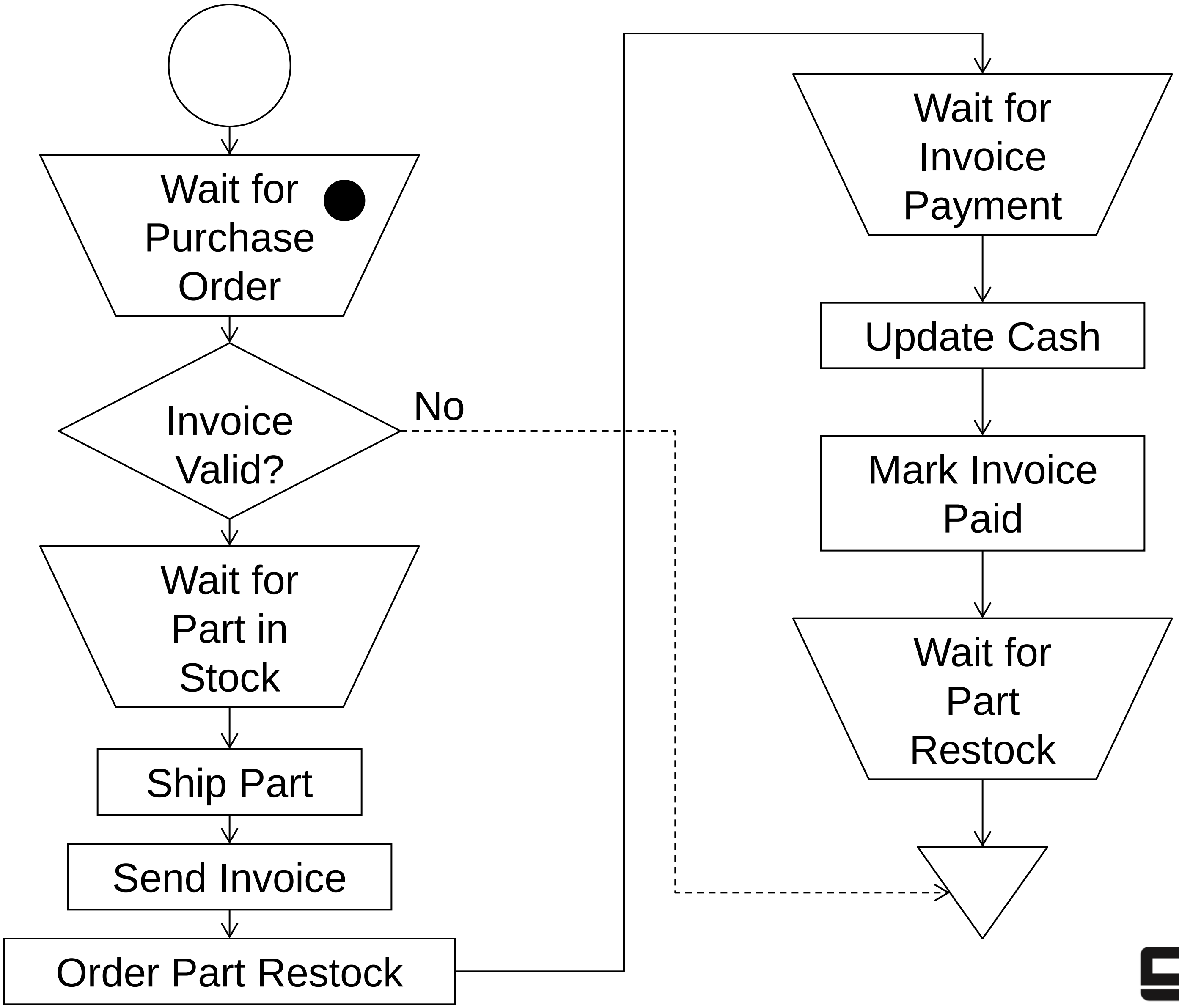
Classic Flowchart (Procedural Rule)



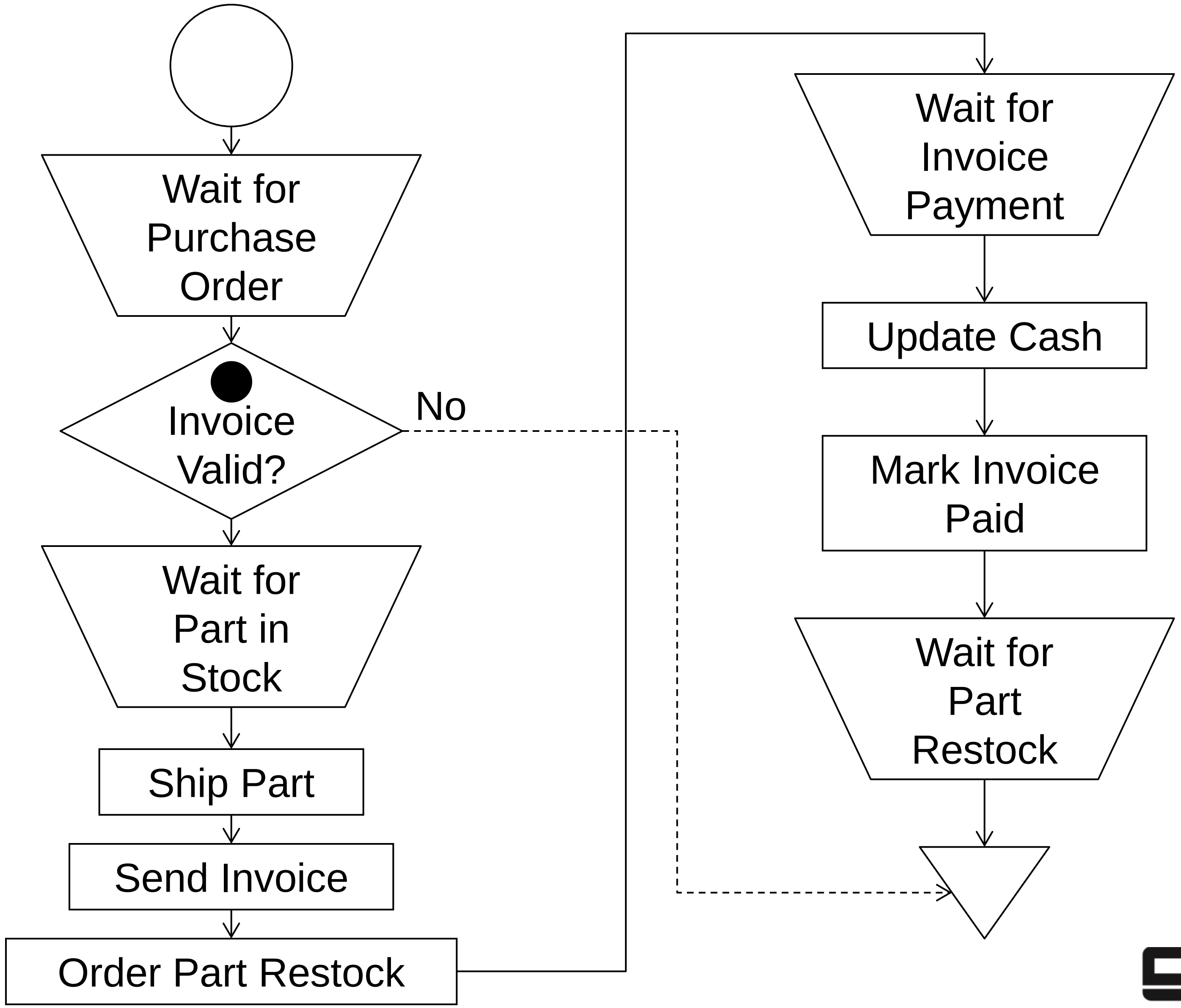
Classic Flowchart Simulation 1



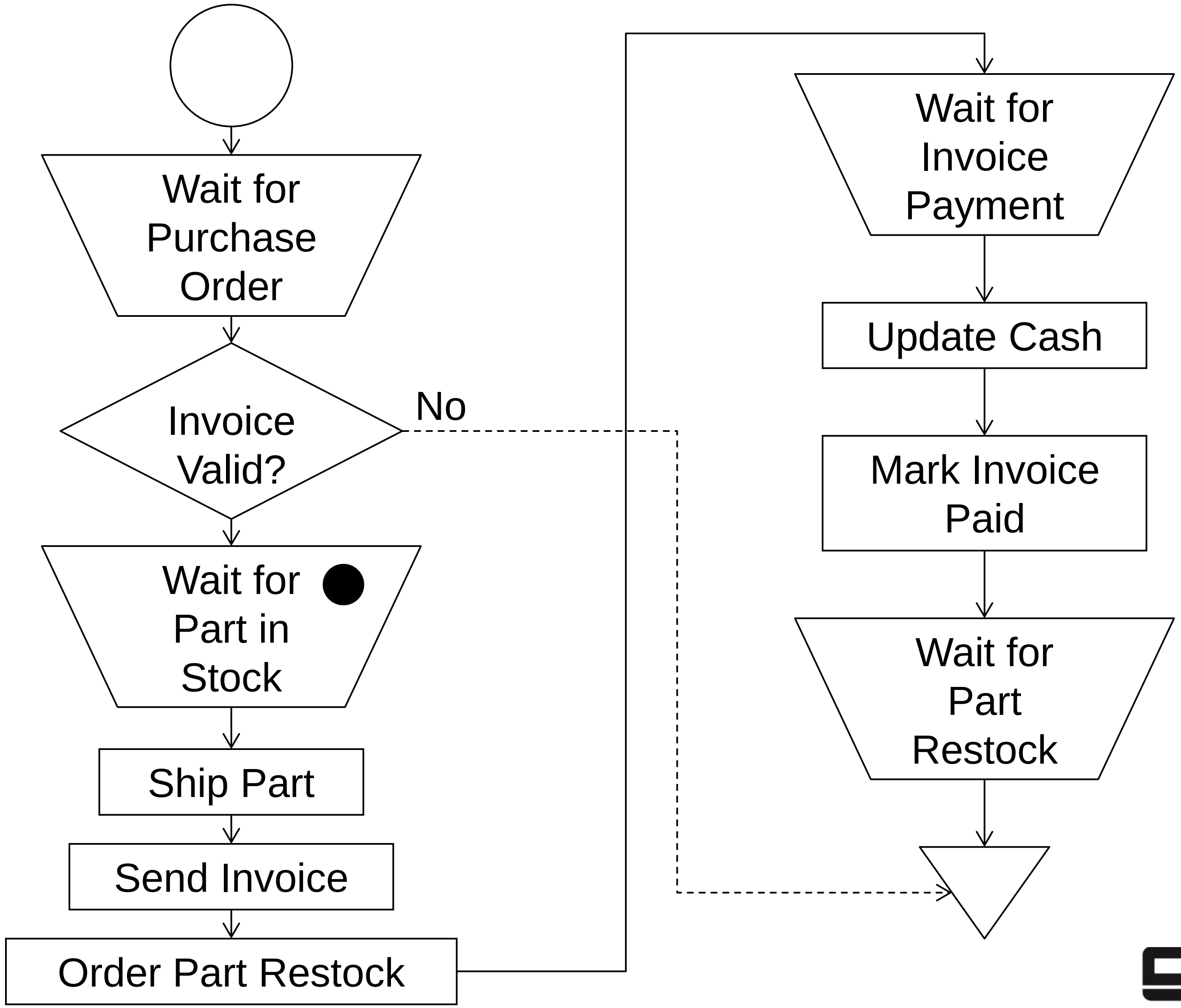
Classic Flowchart Simulation 2



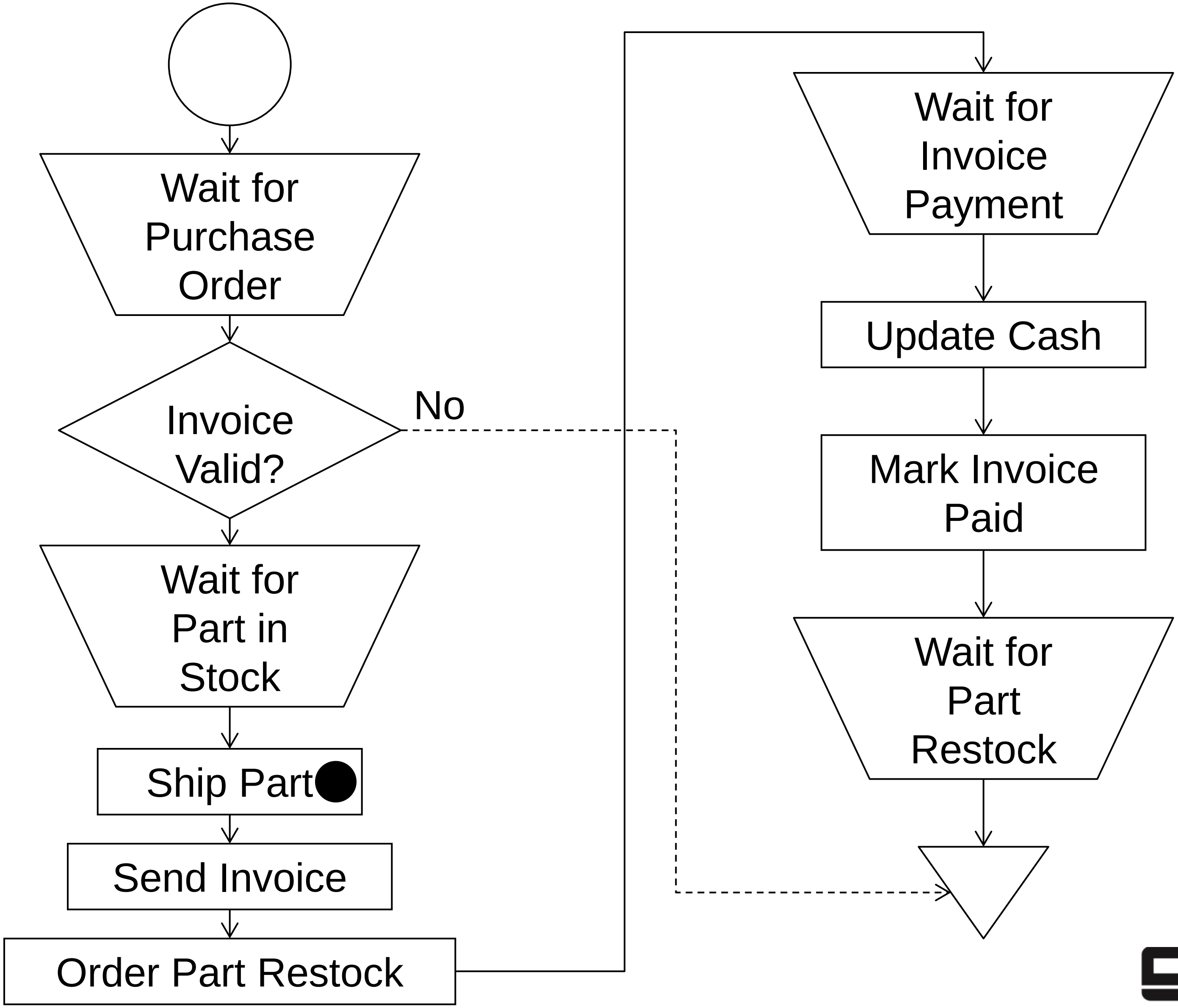
Classic Flowchart Simulation 3



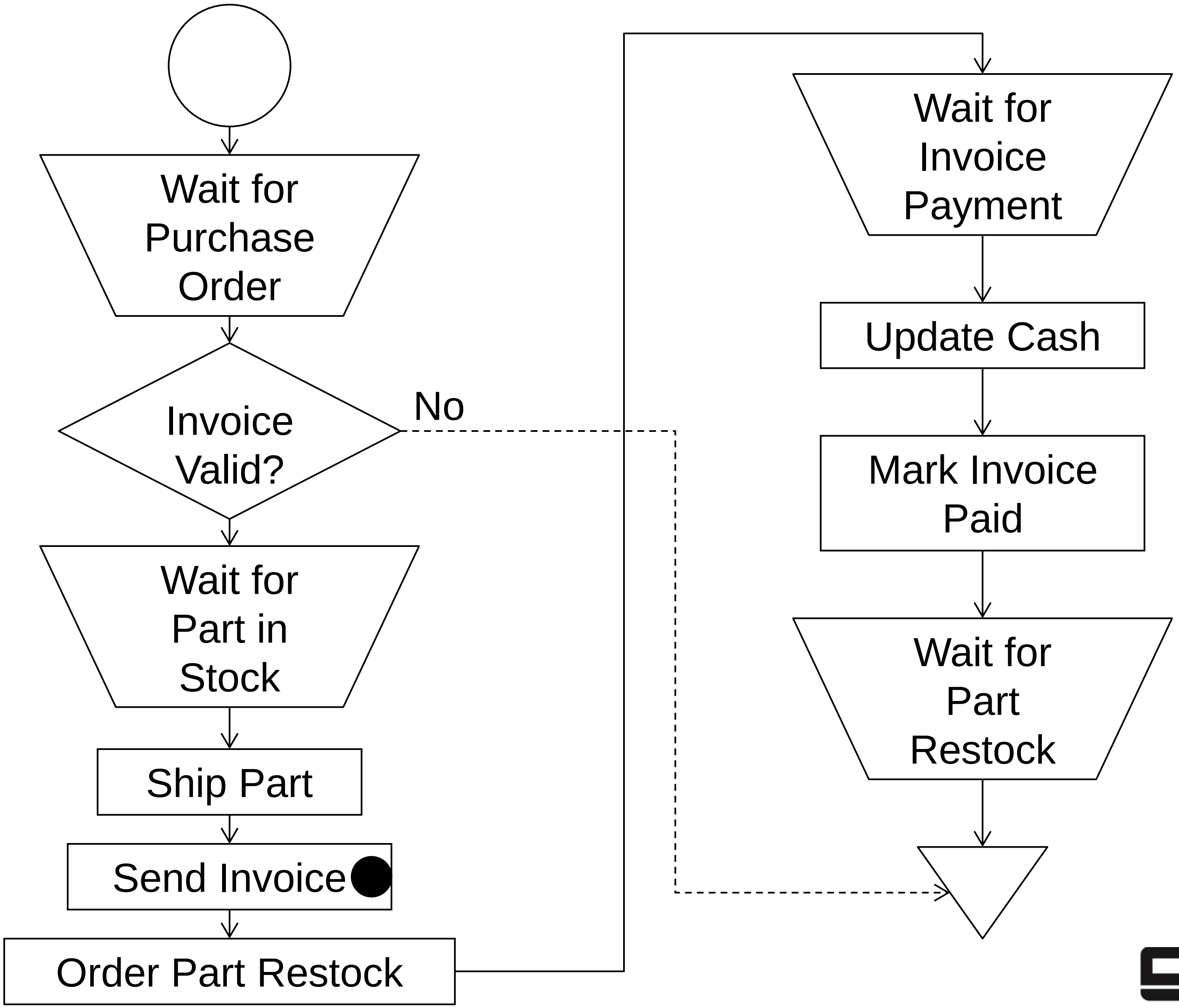
Classic Flowchart Simulation 4



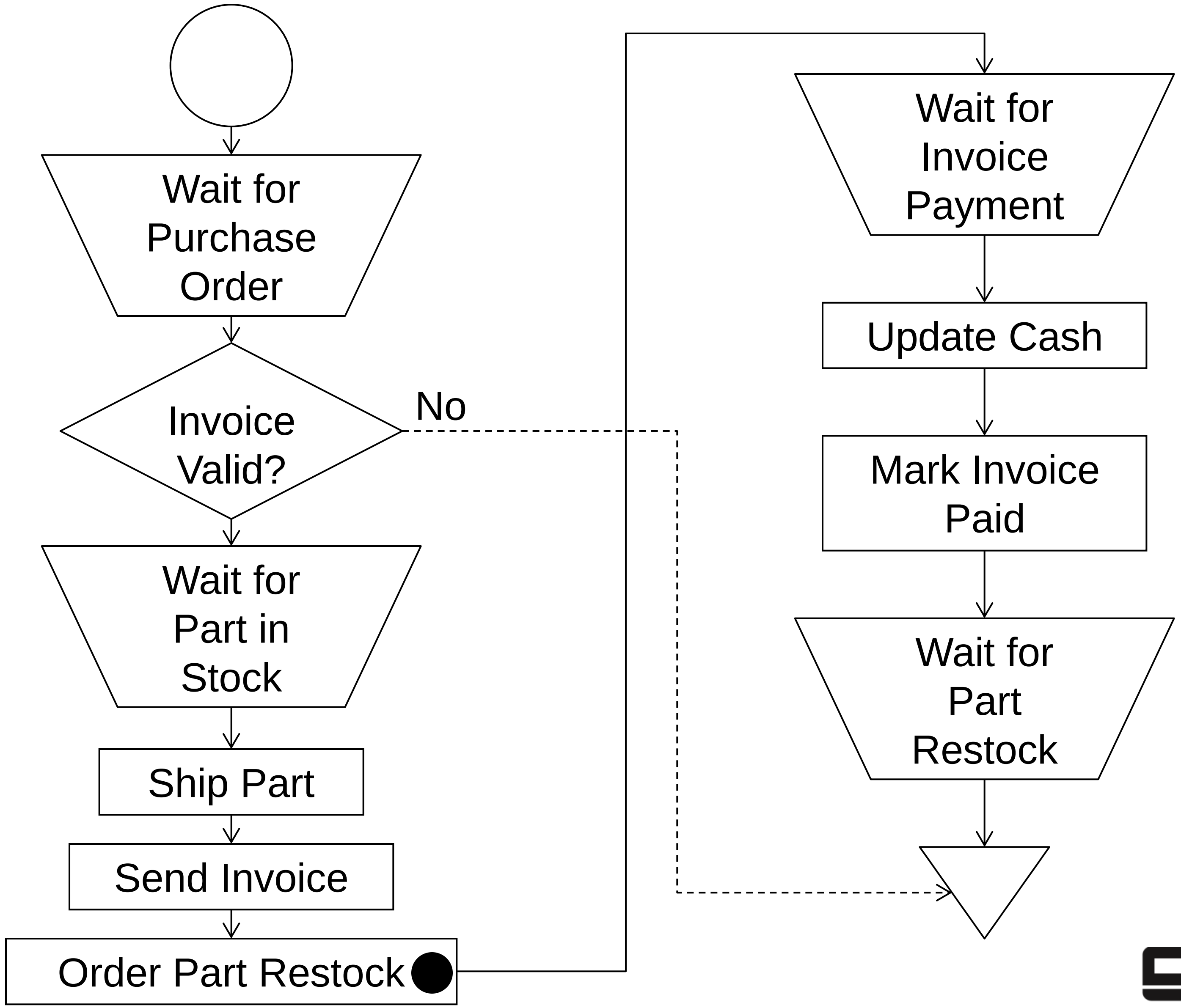
Classic Flowchart Simulation 5



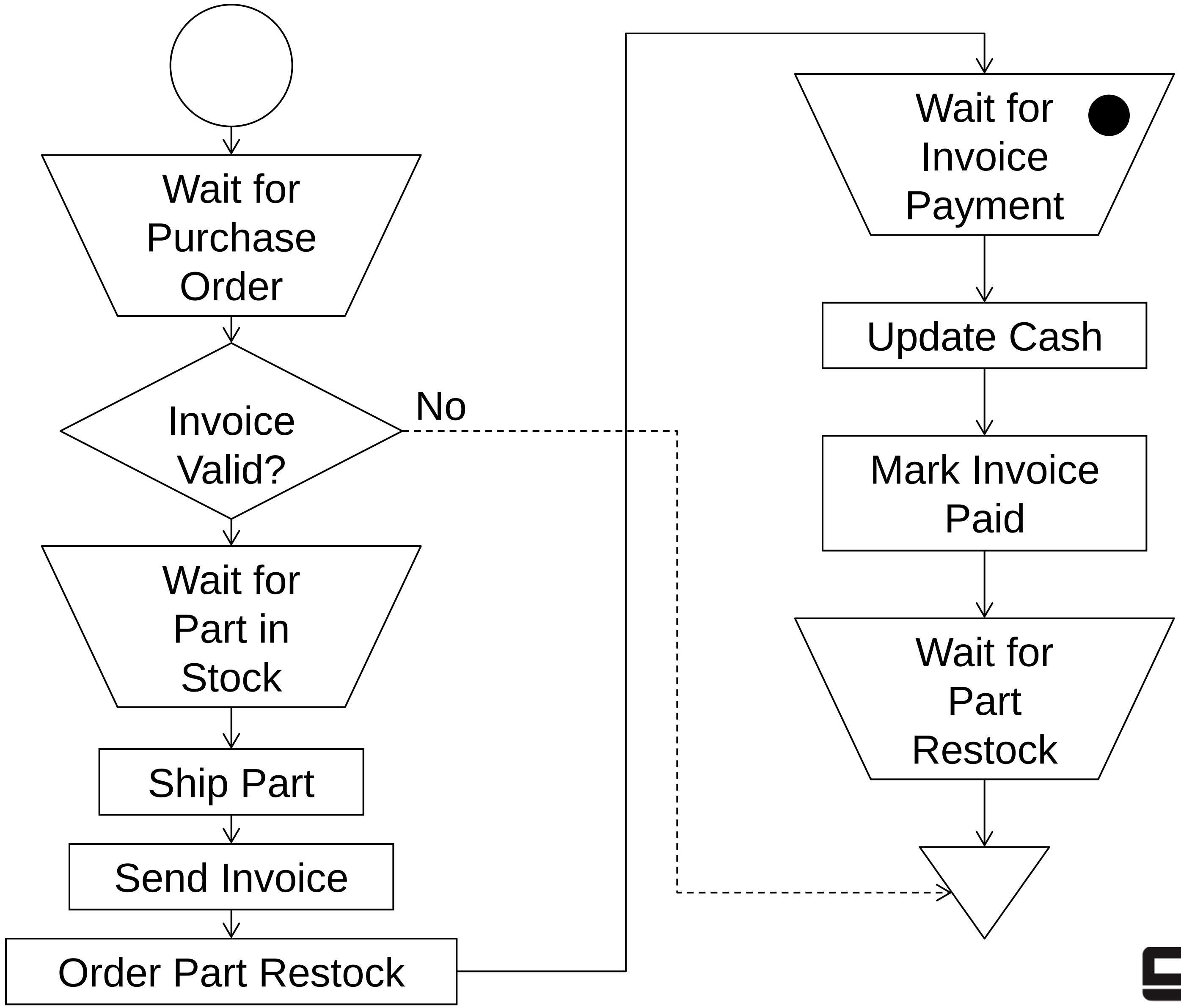
Classic Flowchart Simulation 6



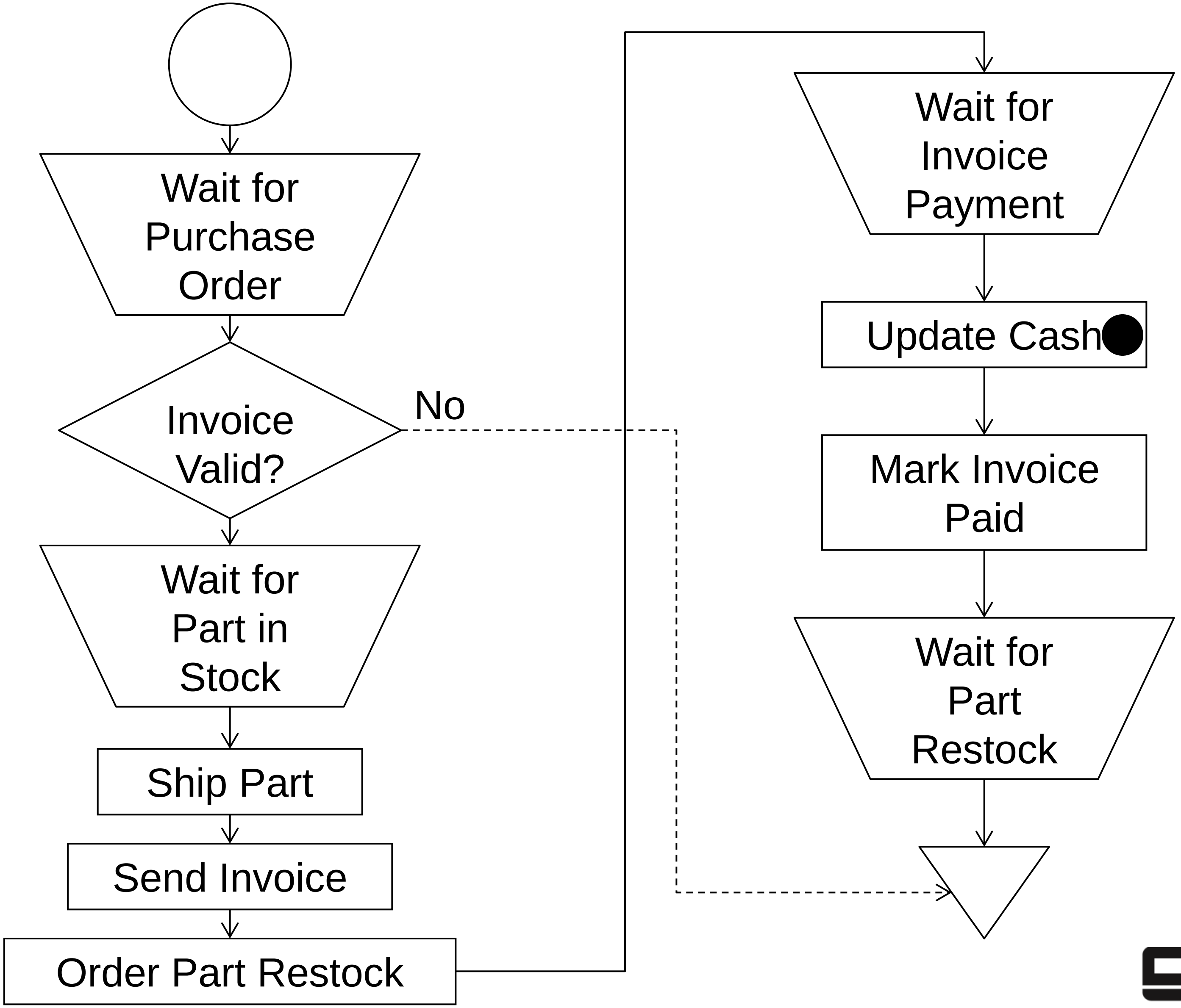
Classic Flowchart Simulation 7



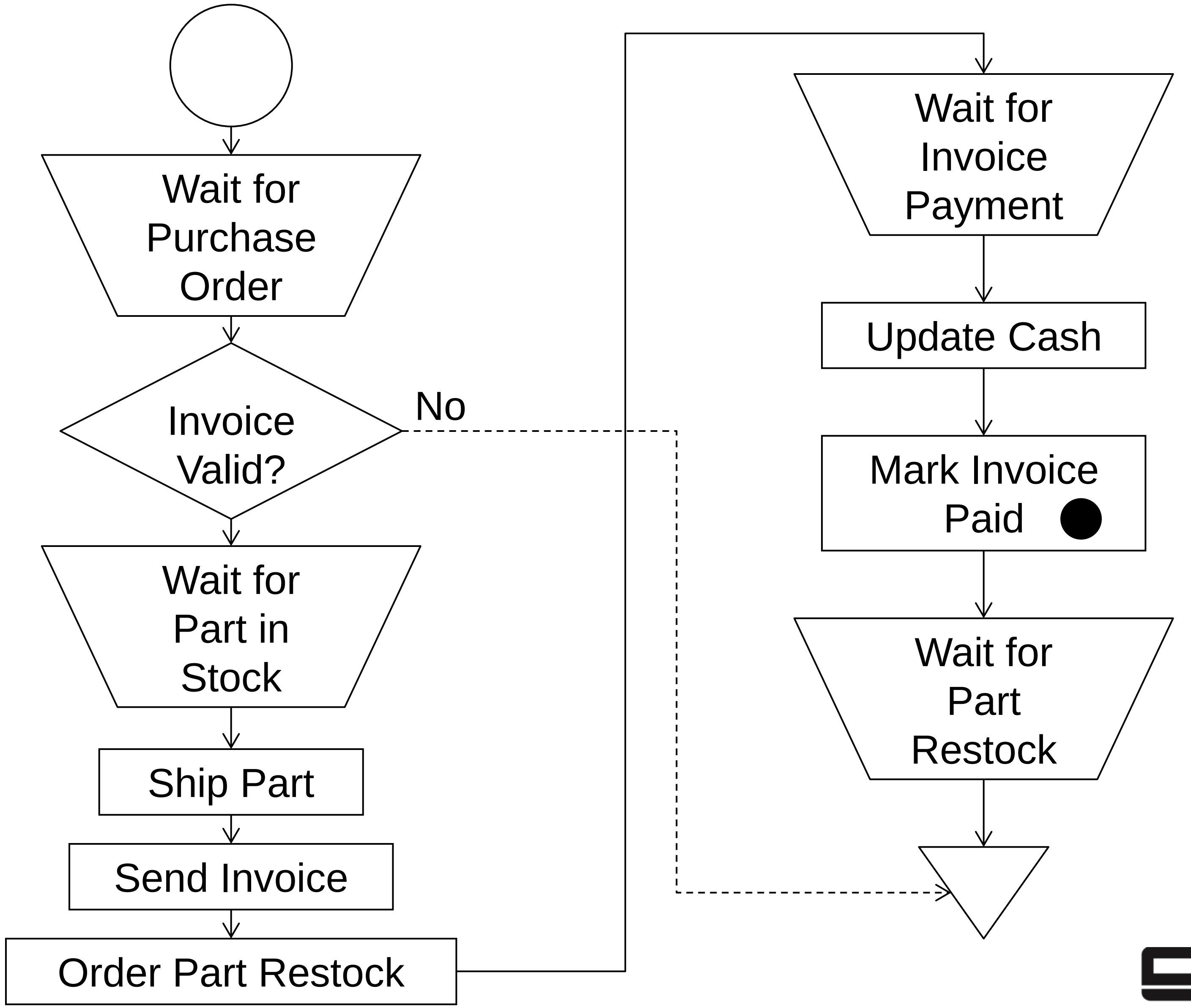
Classic Flowchart Simulation 8



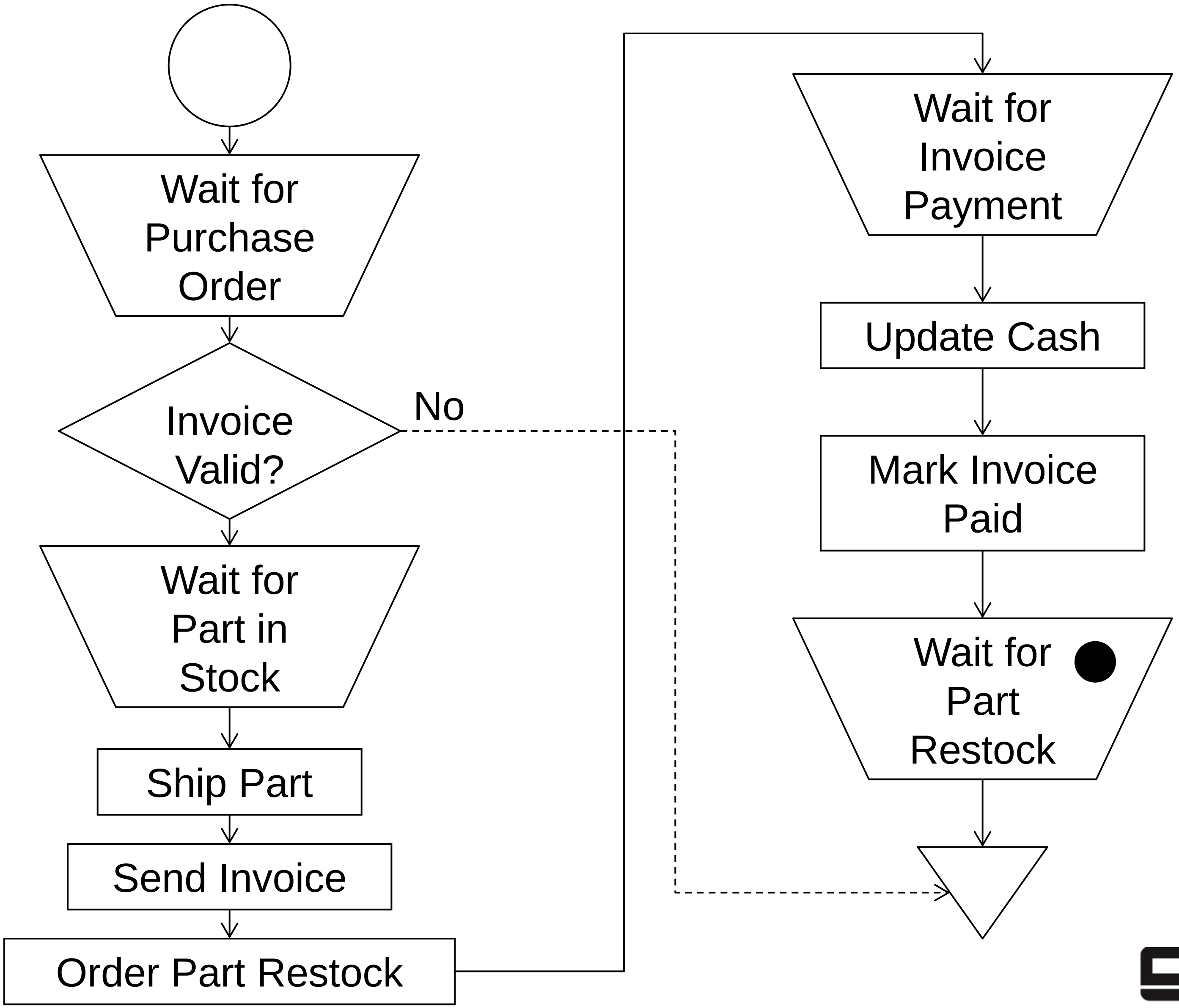
Classic Flowchart Simulation 9



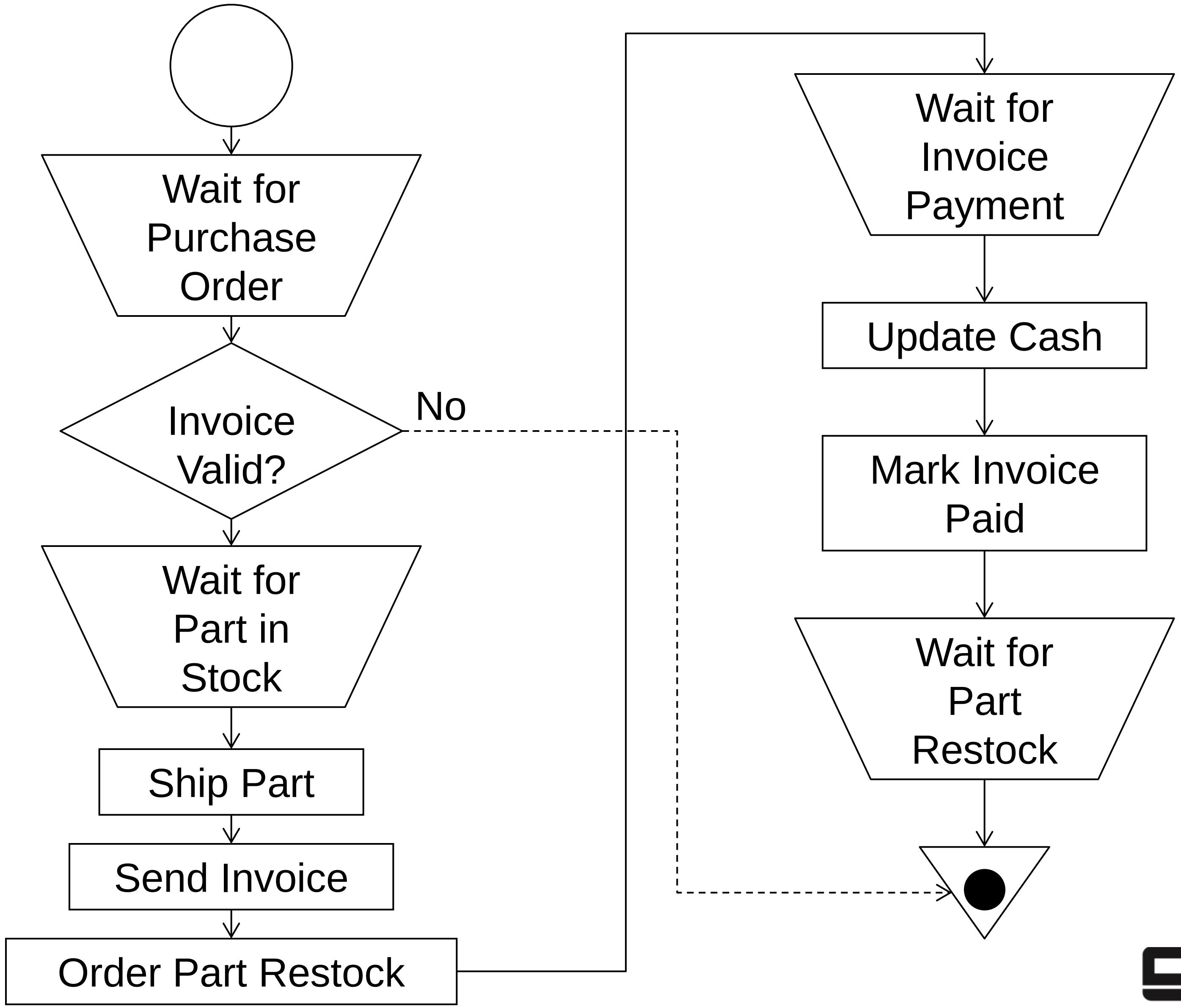
Classic Flowchart Simulation 10



Classic Flowchart Simulation 11



Classic Flowchart Simulation 12



So why not use FlowCharts?

Actions are in terms of… what? *(*“Order Part Restock”*)*

Better if uses abstract, well-defined business actions (`fulfill`)

Overconstrained Sequence of Events

***Why* `Wait for Invoice Payment`**
then* `Wait for Part Restock`*?

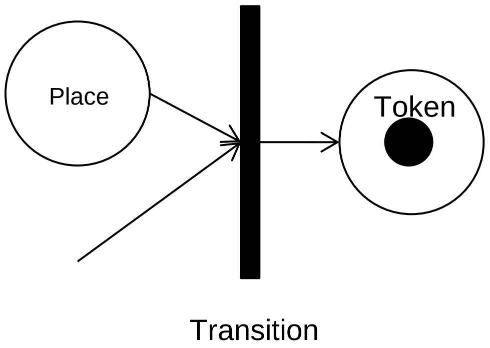
We need more abstract event sequencing

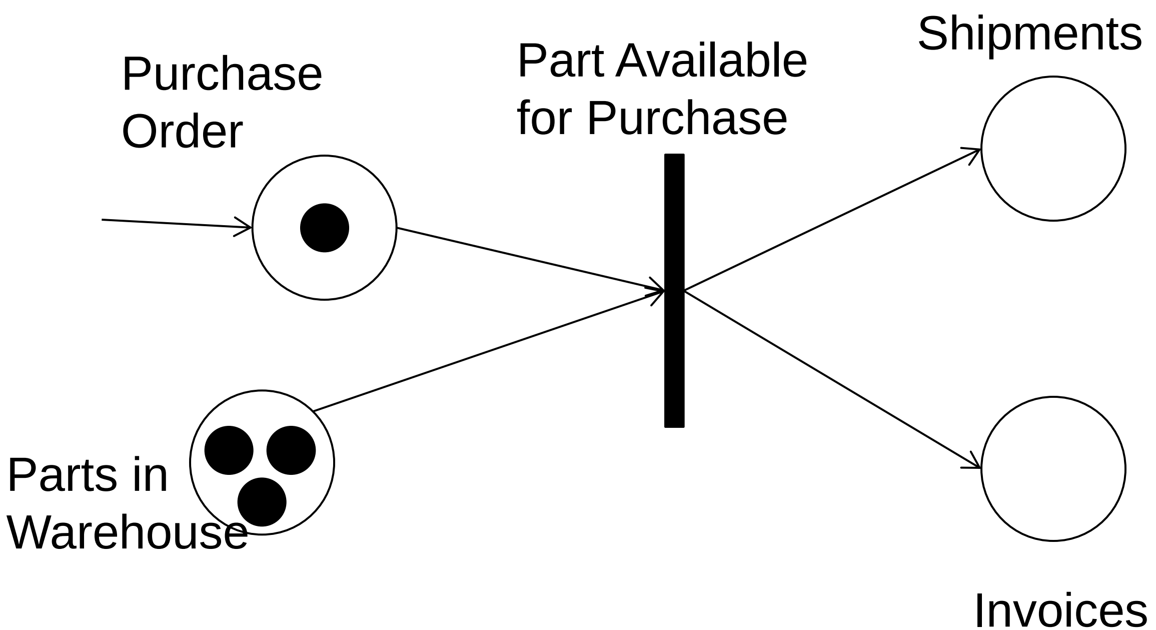
Synchronization is not handled well

What does “wait for …” mean?

Classic Petri Net 1
Synchronization with Multiple Tokens

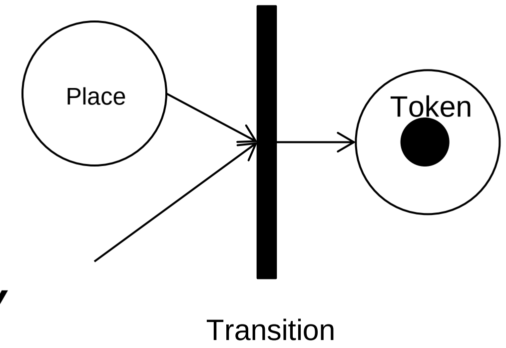
Place:* *A holder of zero or more (black) tokens
Token:* *Marker in place representing state-is-active
Outgoing Arc:* *Connection from an (input) Place to Transition
Incoming Arc:* *Connection from Transition to (Output) Place
Transition:* *An intermediary between places that consumes tokens from input places synchronously and generates a token in output places

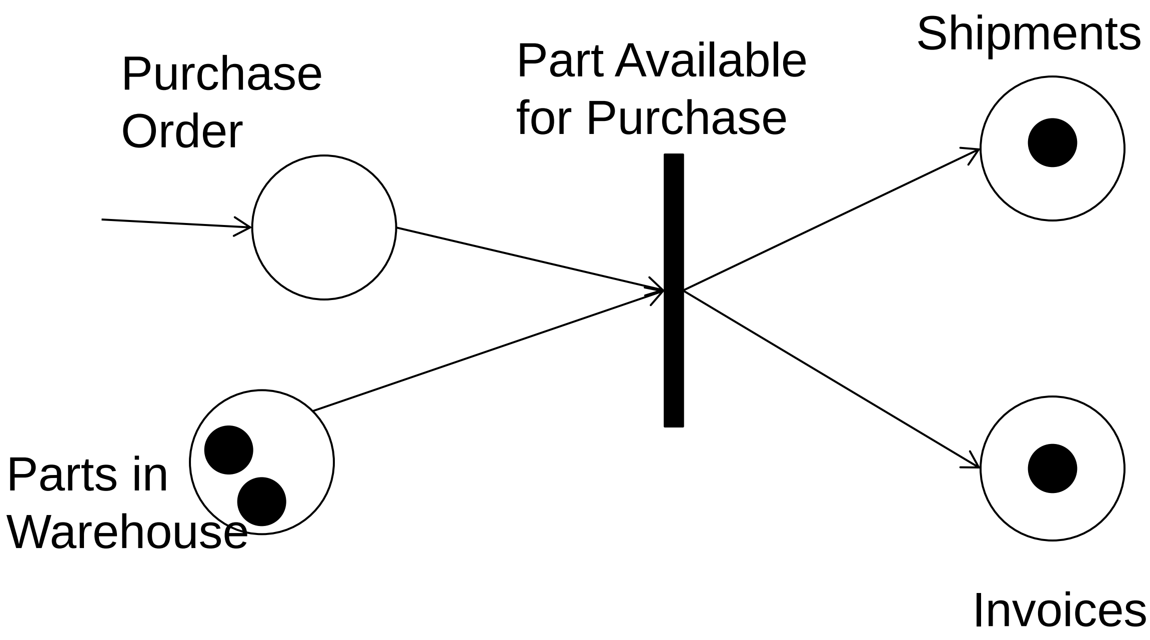




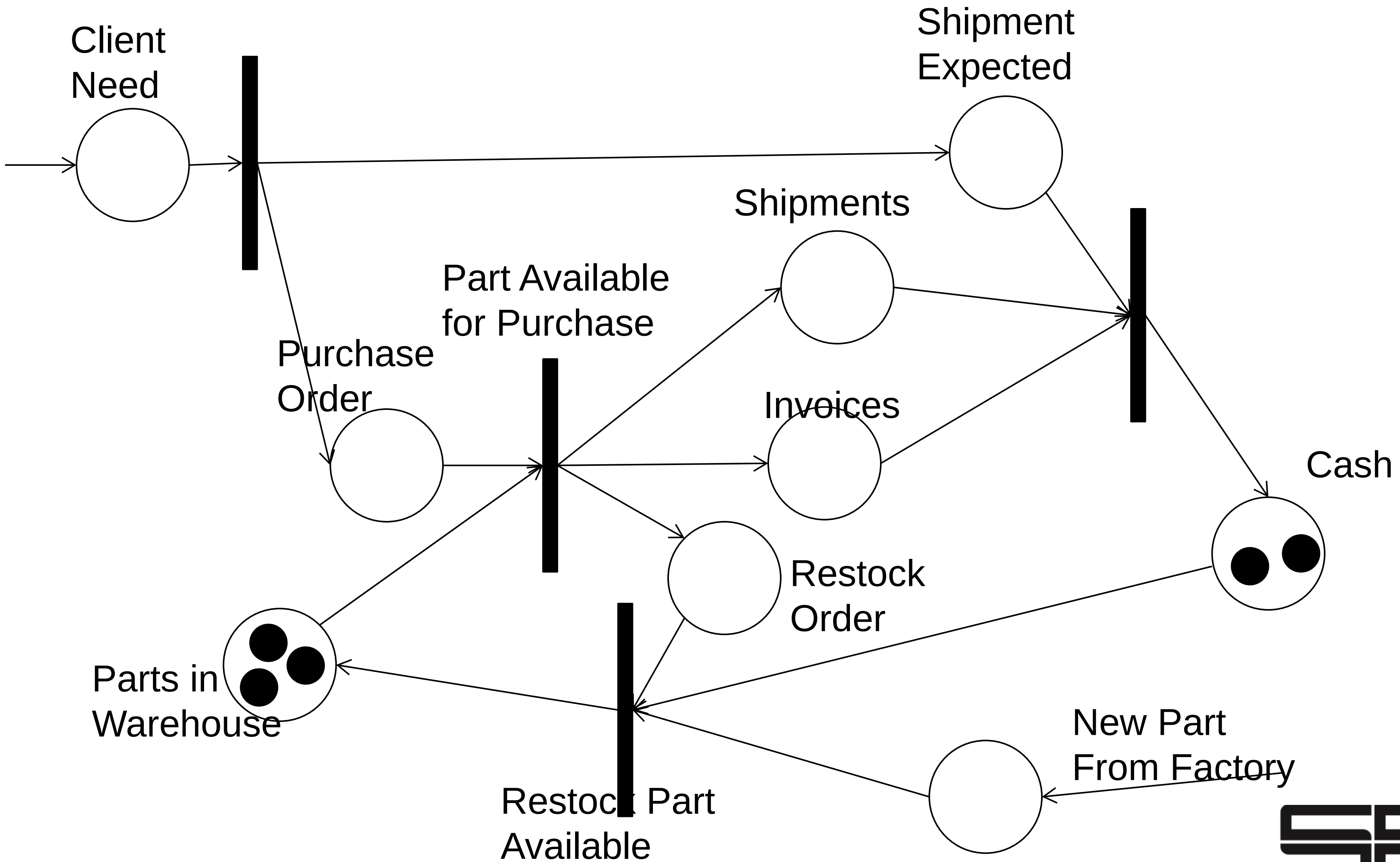
Classic Petri Net 2
Synchronization with Multiple Tokens

***Place:** A holder of zero or more (black) tokens*
***Token:** Marker in place representing state-is-active*
***Outgoing Arc:** Connection from an (input) Place to Transition*
***Incoming Arc:** Connection from Transition to (Output) Place*
***Transition:** An intermediary between places that consumes tokens from input places synchronously and generates a token in output places*

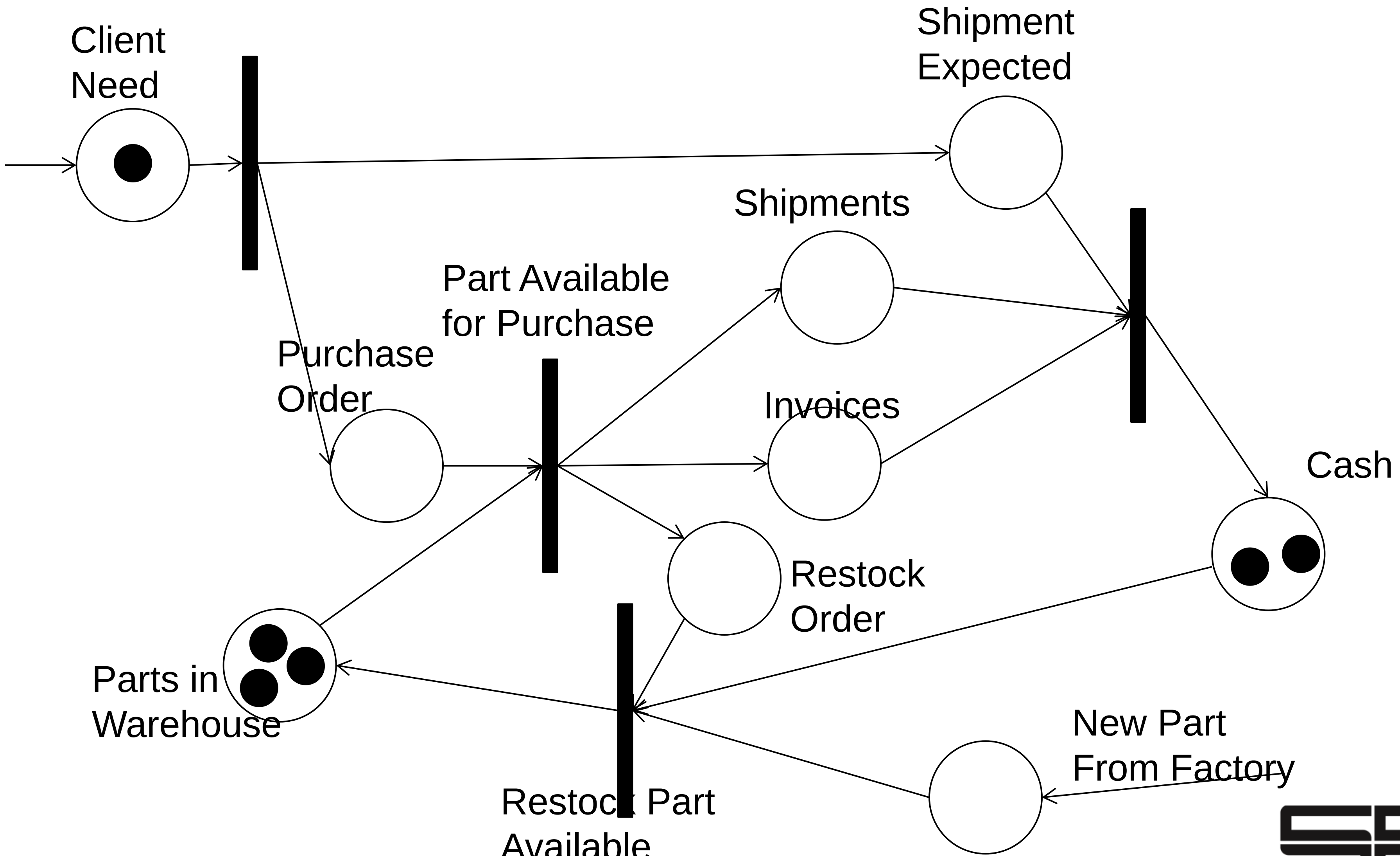




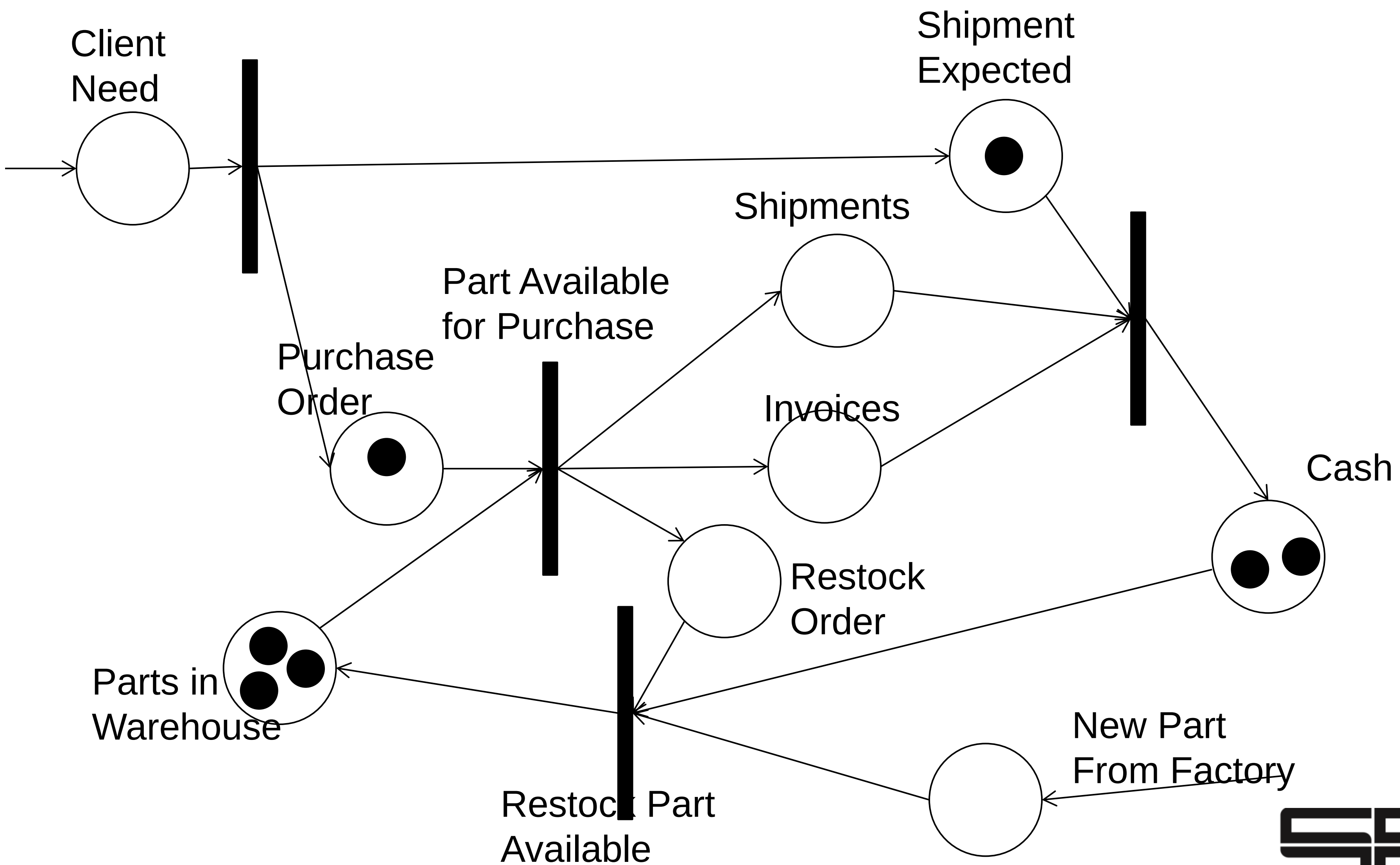
Classic Petri Net Simulation 1



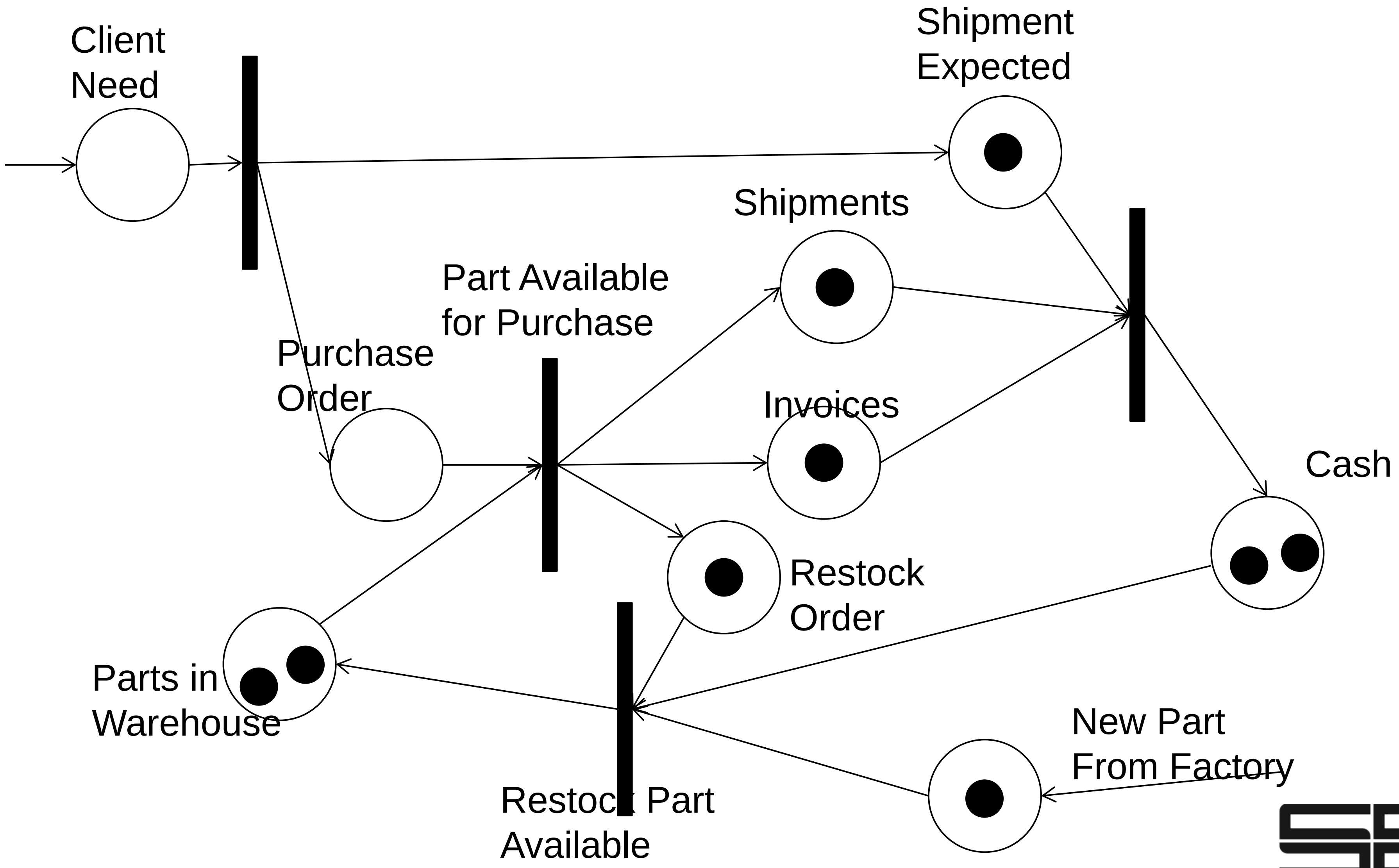
Classic Petri Net Simulation 2



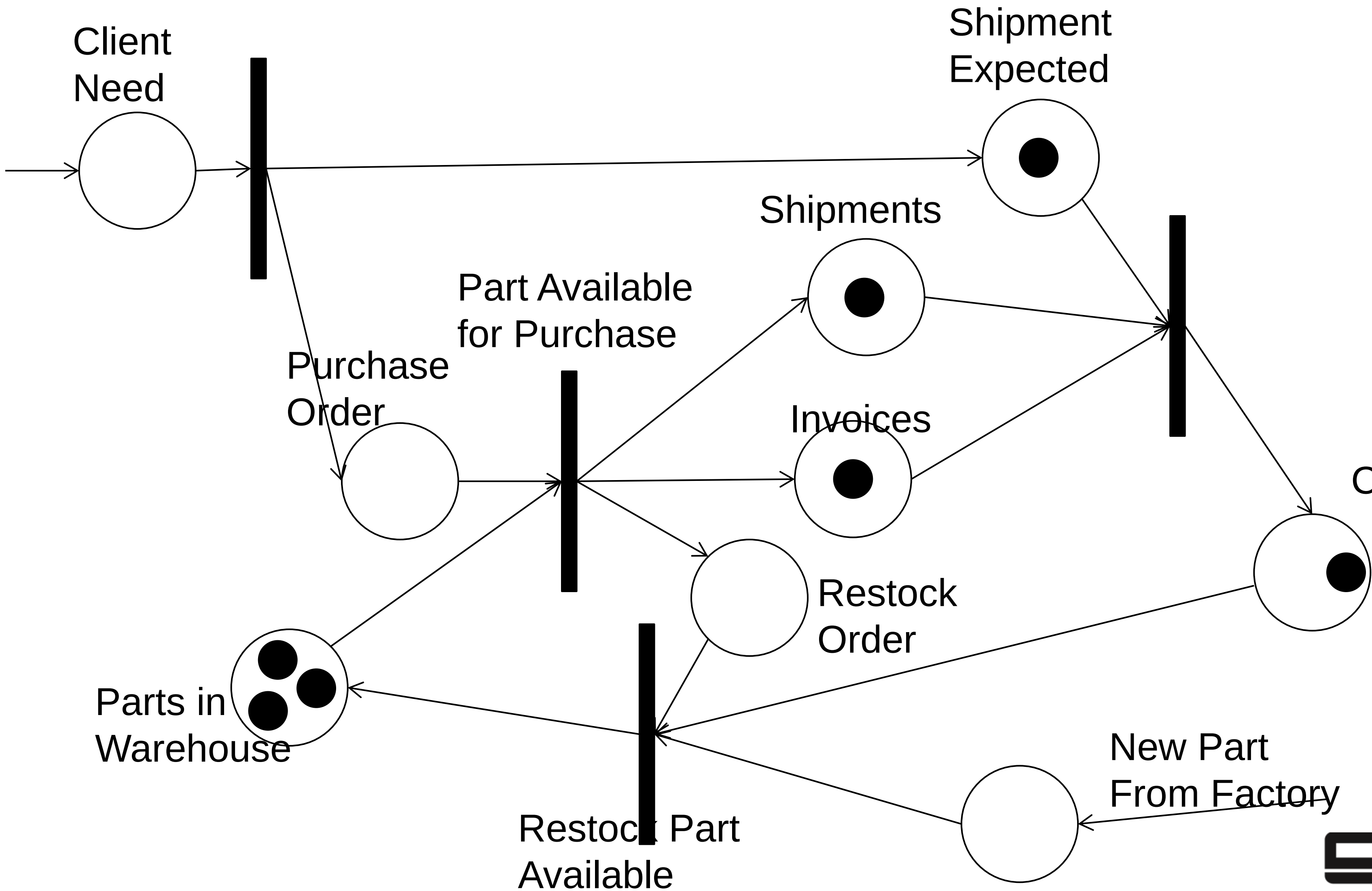
Classic Petri Net Simulation 3



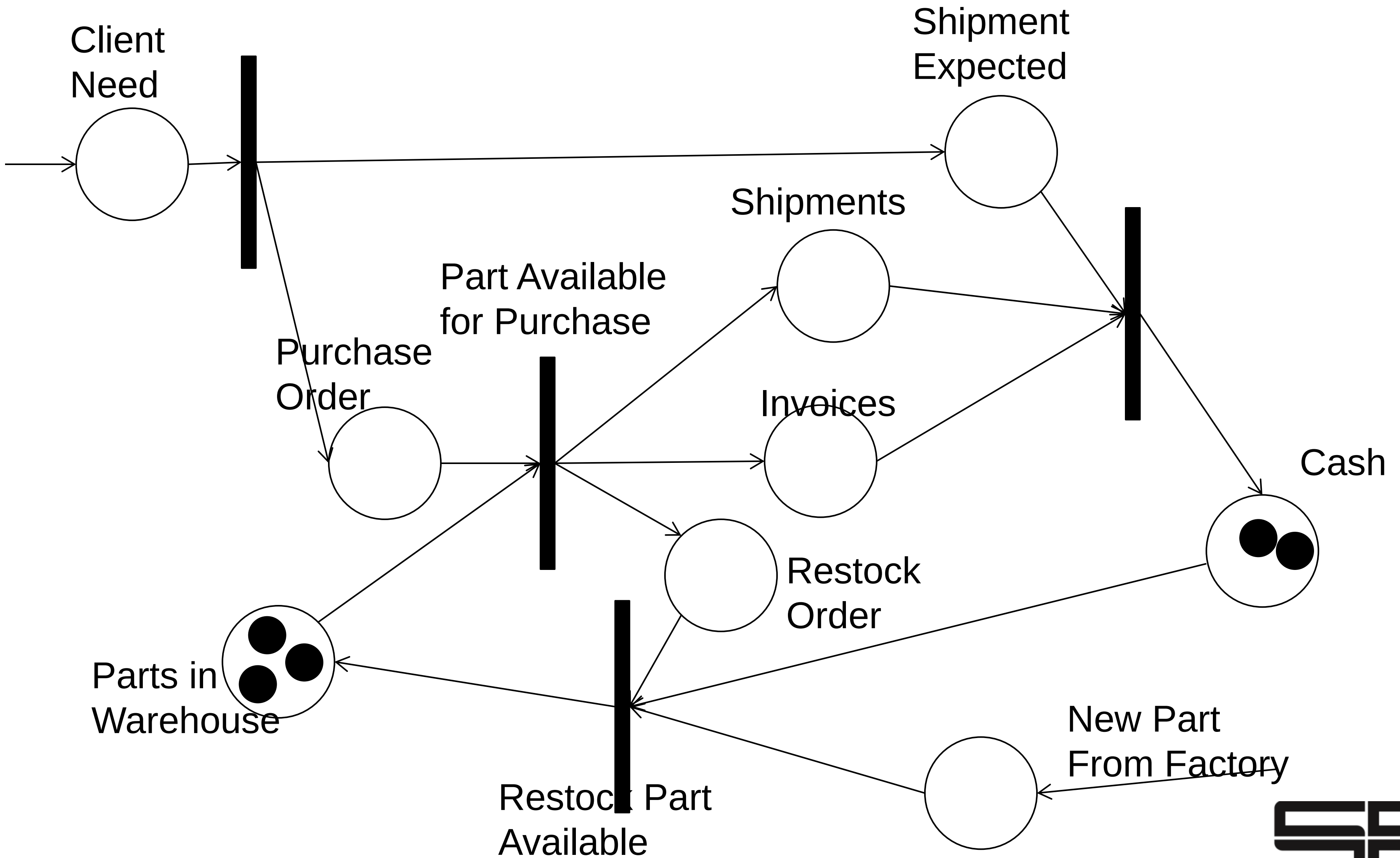
Classic Petri Net Simulation 4



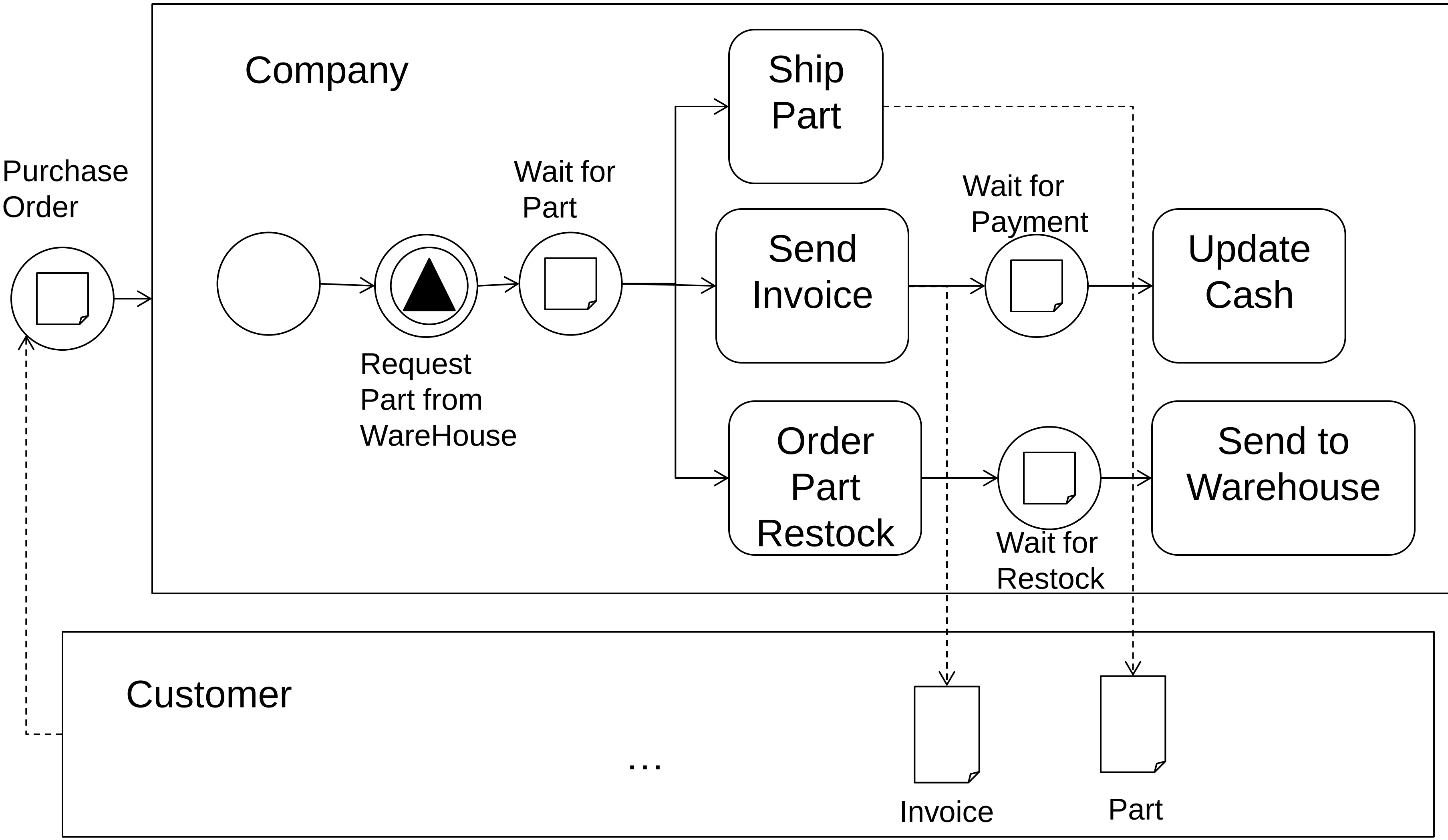
Classic Petri Net Simulation 5



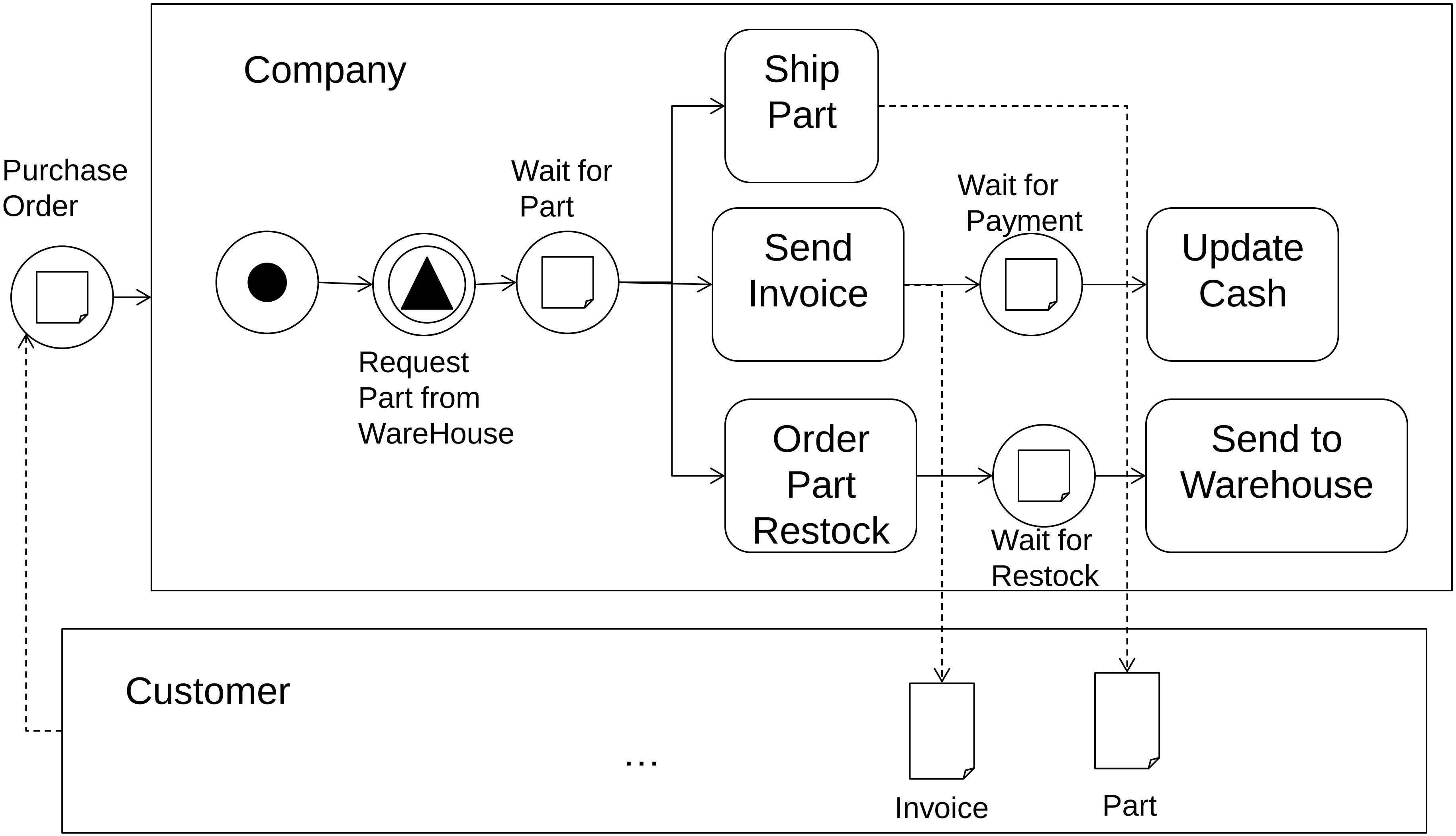
Classic Petri Net Simulation 6



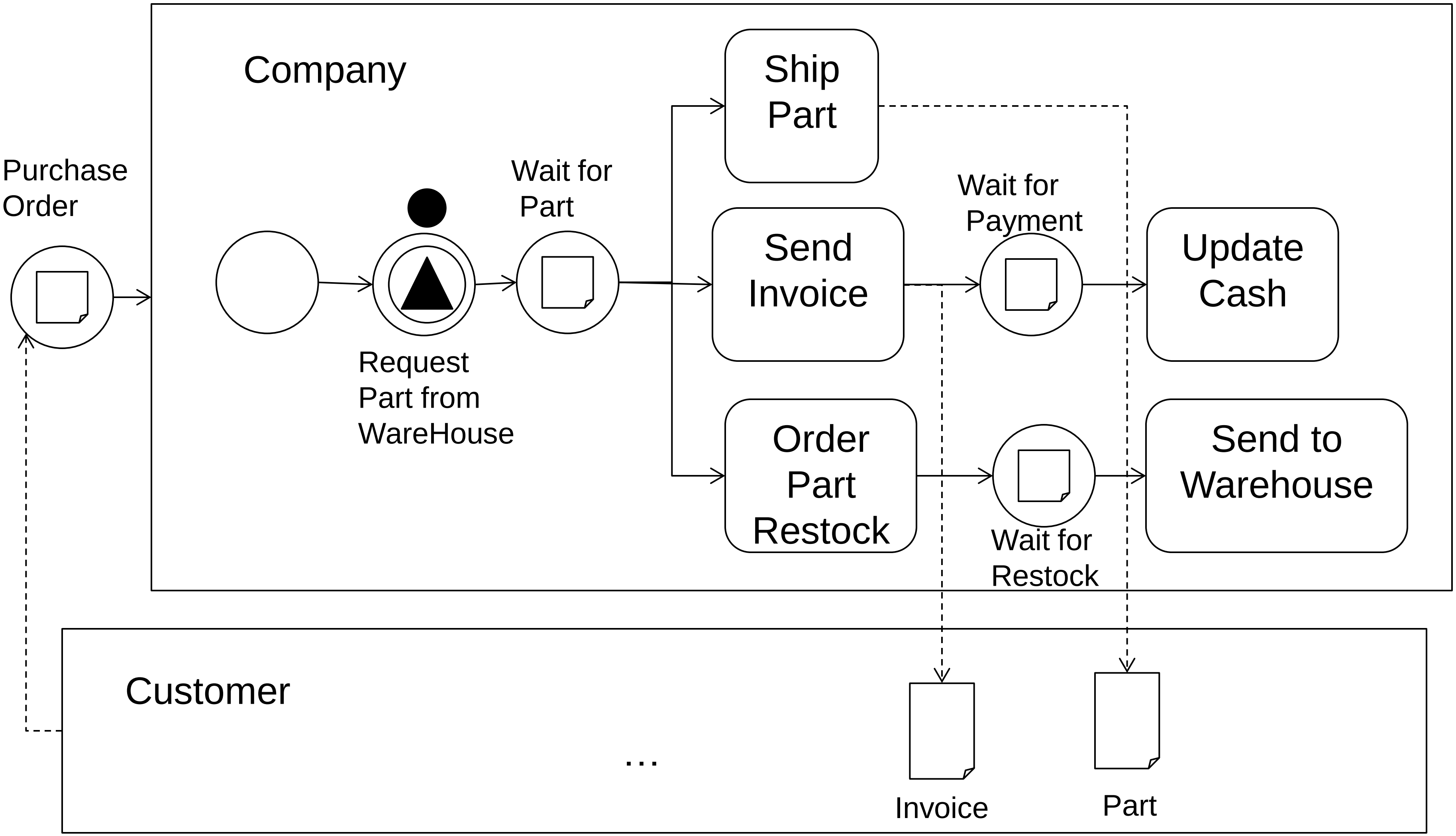
BPMN: A special kind of (classic) Petri Net



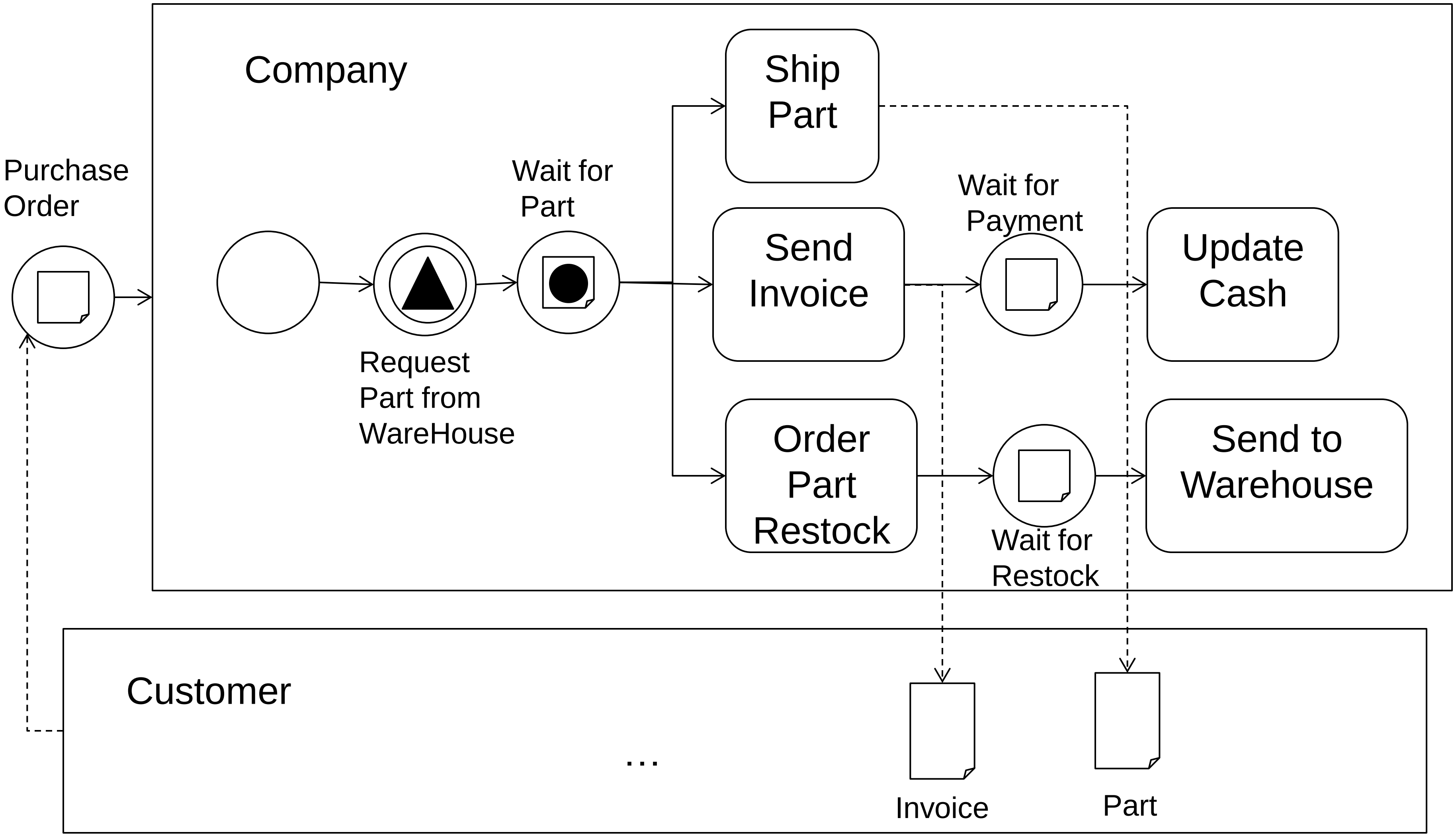
BPMN Simulation 1



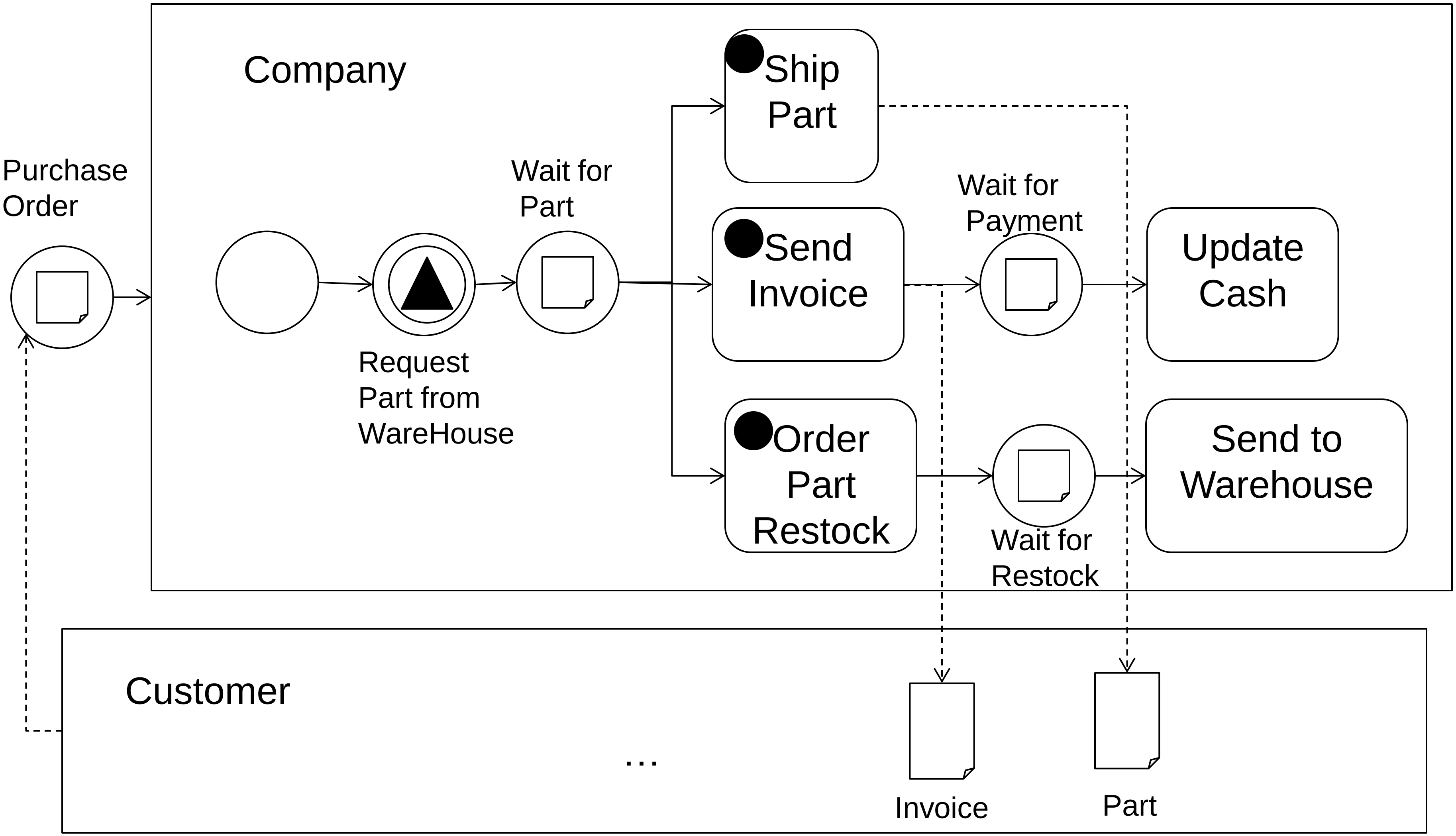
BPMN Simulation 2



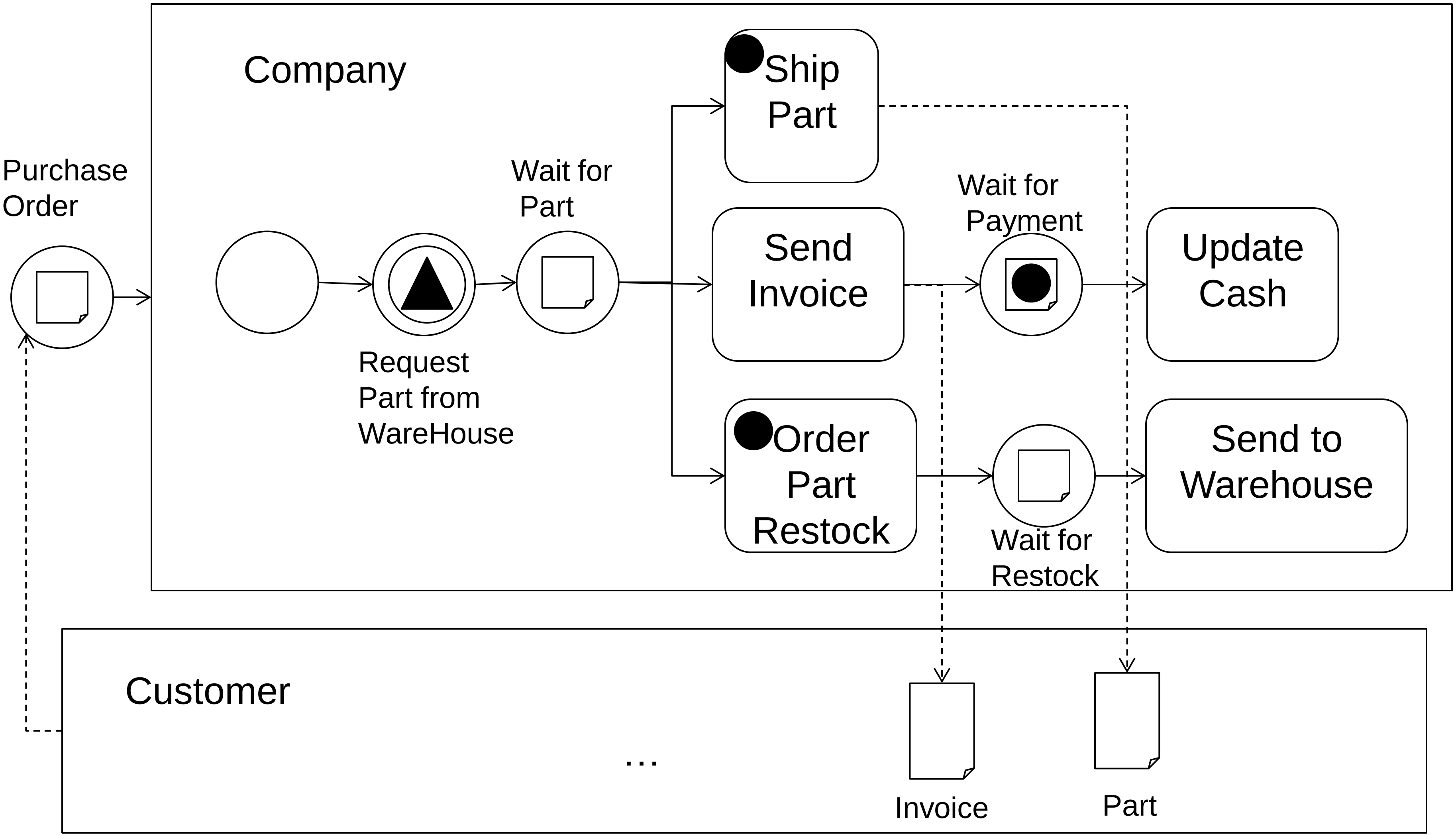
BPMN Simulation 3



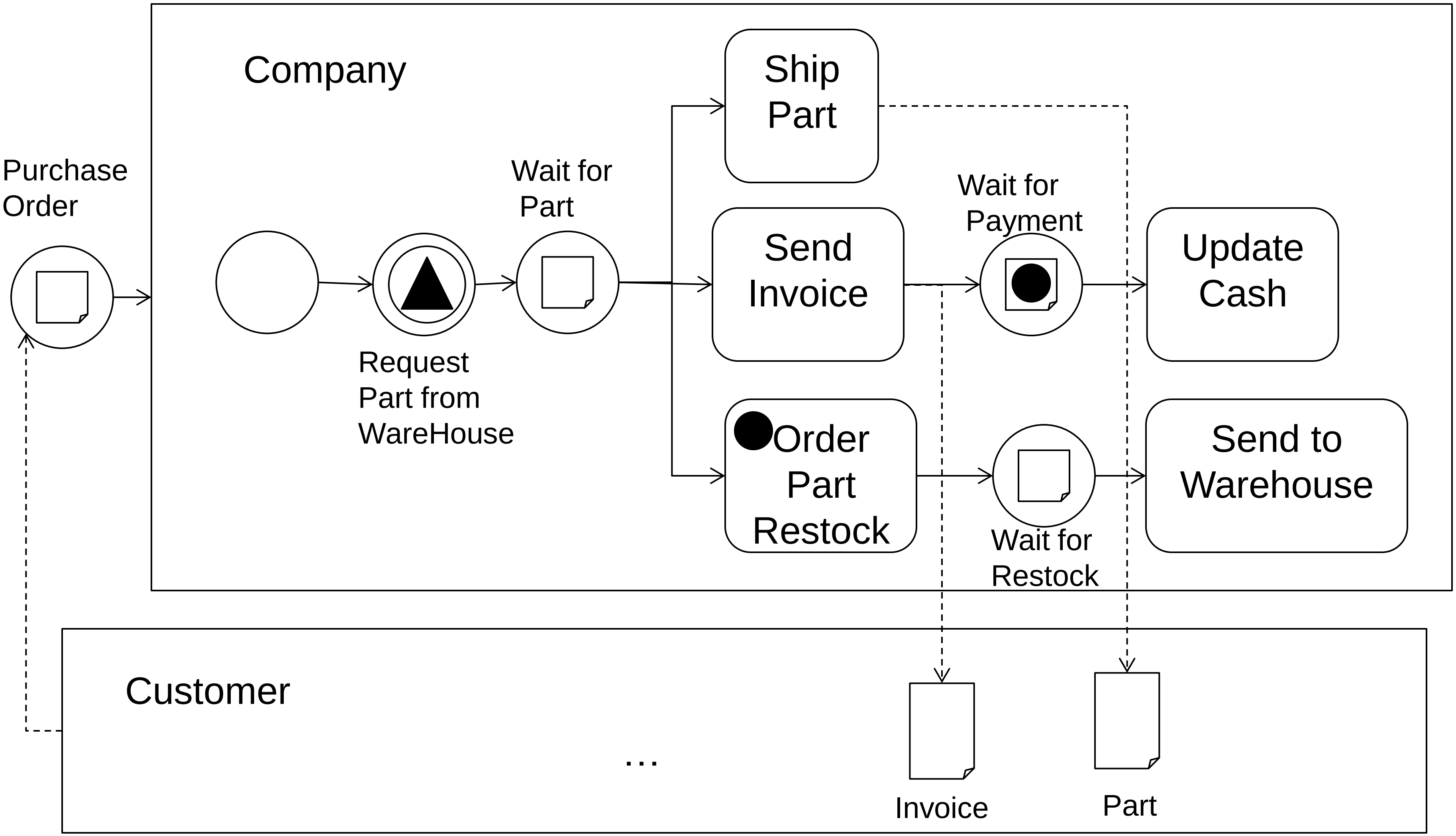
BPMN Simulation 4



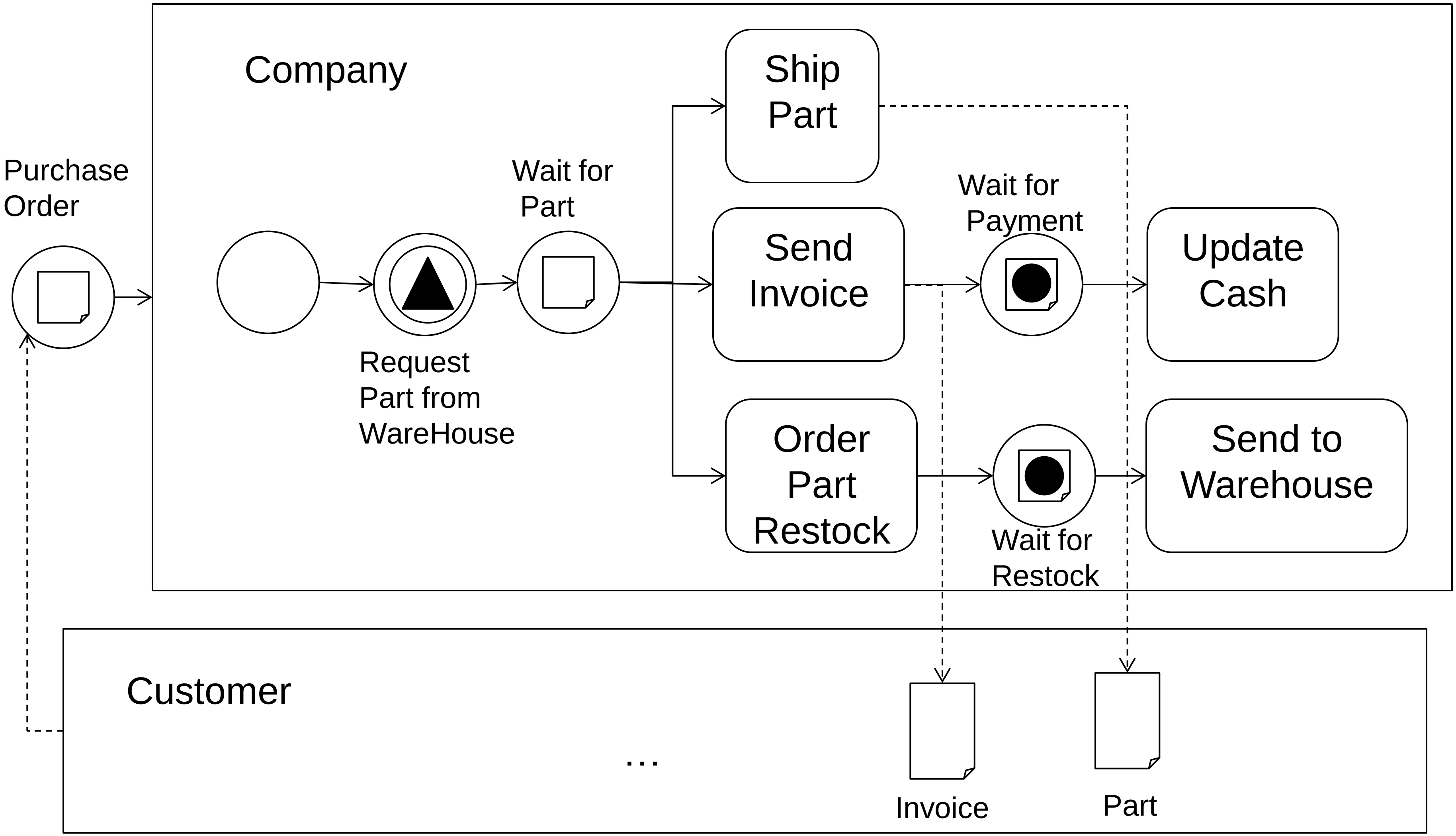
BPMN Simulation 5



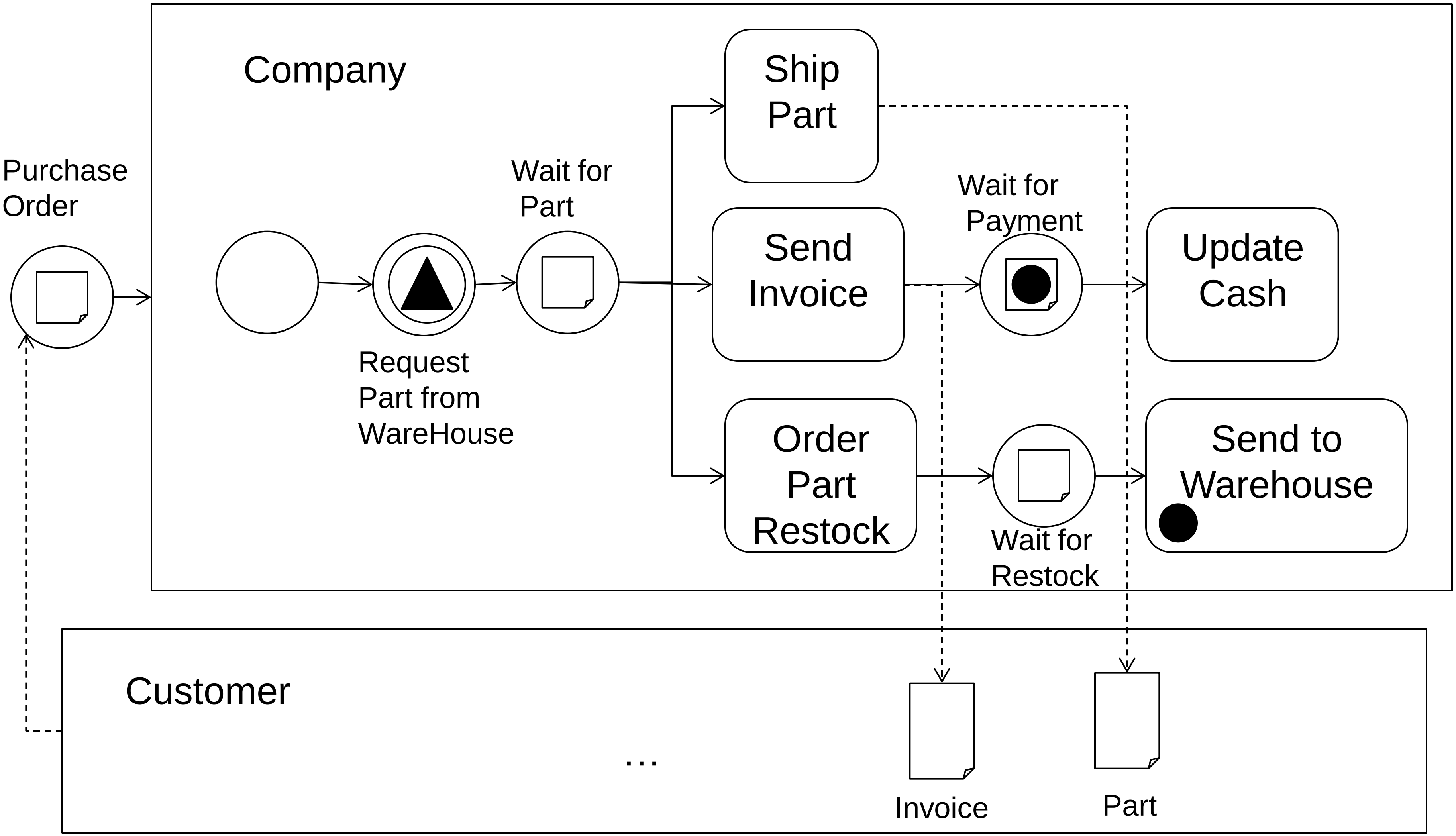
BPMN Simulation 6



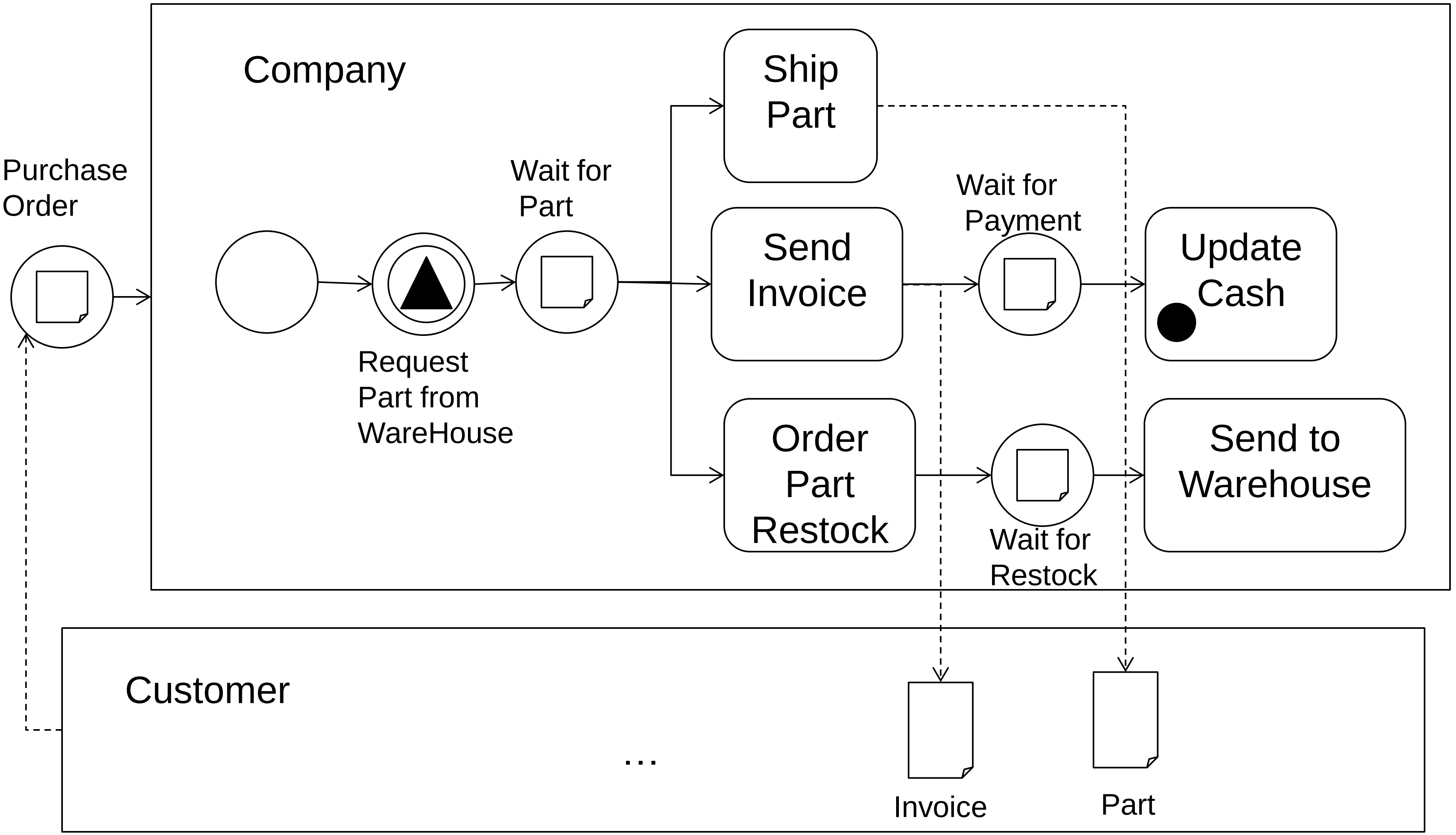
BPMN Simulation 7



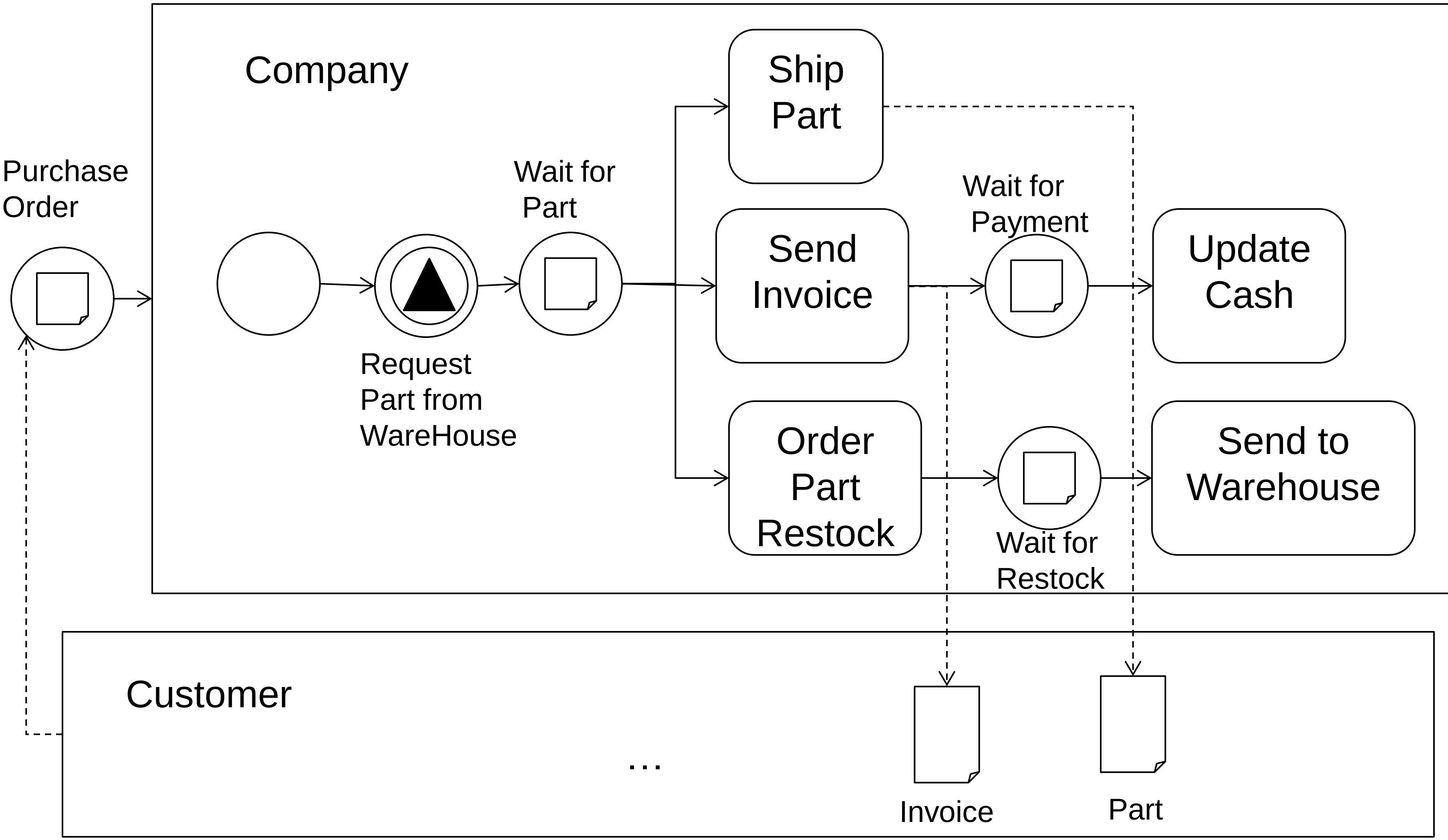
BPMN Simulation 8



BPMN Simulation 9



BPMN Simulation 10



Weaknesses of BPMN?

Actions are in terms of… what? *(*“Order Part Restock”*)*

Better if uses abstract, well-defined business actions (`fulfill`)

Data being exchanged isn’t clearly defined

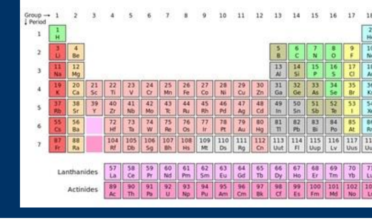
Better if uses abstract, well-defined business data (`PurchaseOrder`

Synchronization often occurs with data transmission

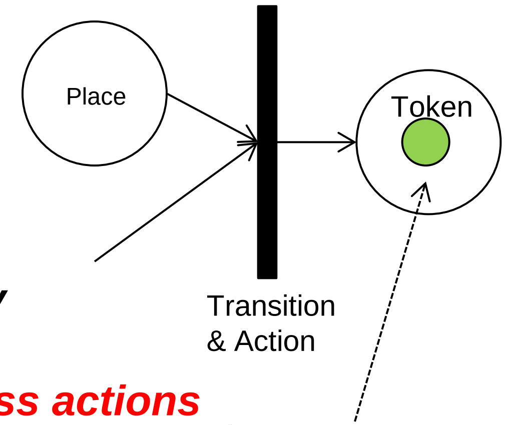
BPMN requires (active) event and (passive) data

Better to combine synchronization signals with data

“Colored” Petri Nets (Timed, Hierarchical, …)

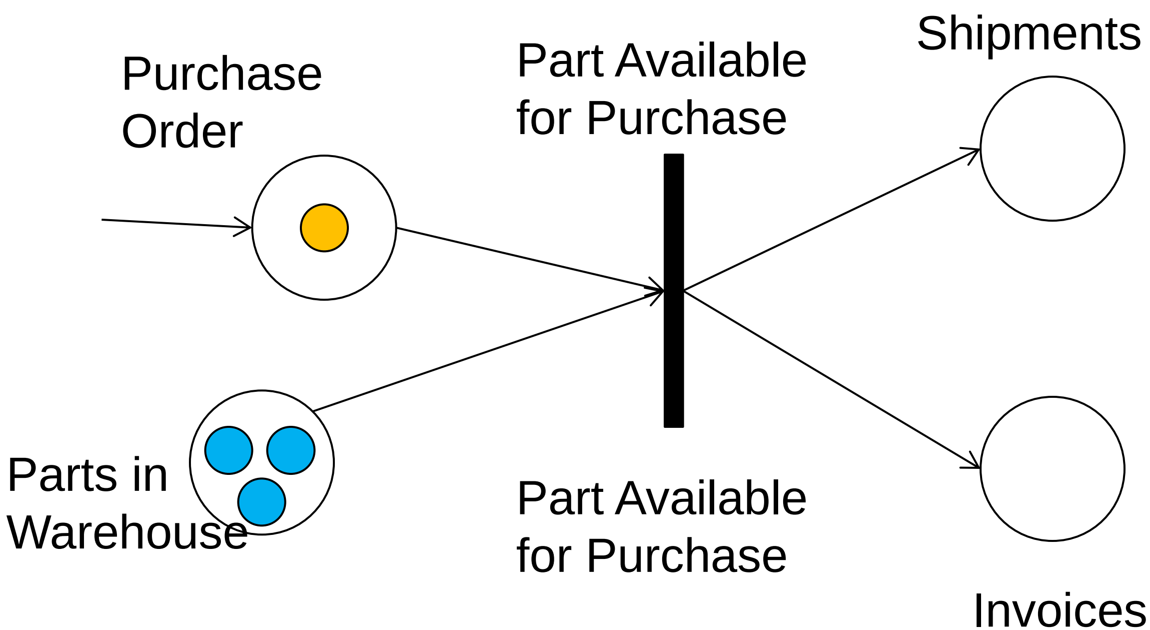


***Place:* A holder of zero or more (colored) tokens**
***Token:* Marker in place representing available event with data**
***Outgoing Arc:* Connection from an (input) Place to Transition**
***Incoming Arc:* Connection from Transition to (Output) Place**
***Transition:* An intermediary between places**
that consumes tokens from input places synchronously
and generates a token in output places
Computes new data values to new tokens using business actions

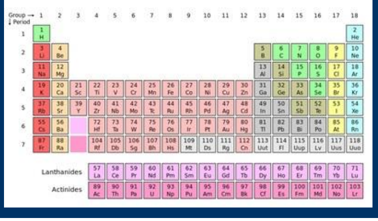


Token color represents abstract business data

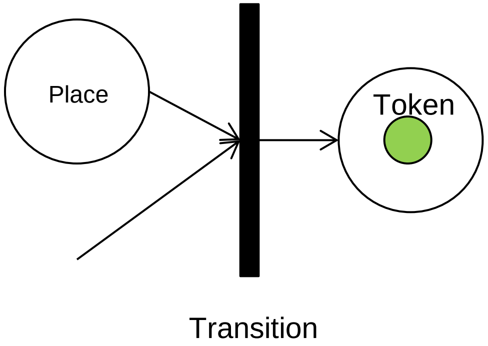
(Can use name instead of color)

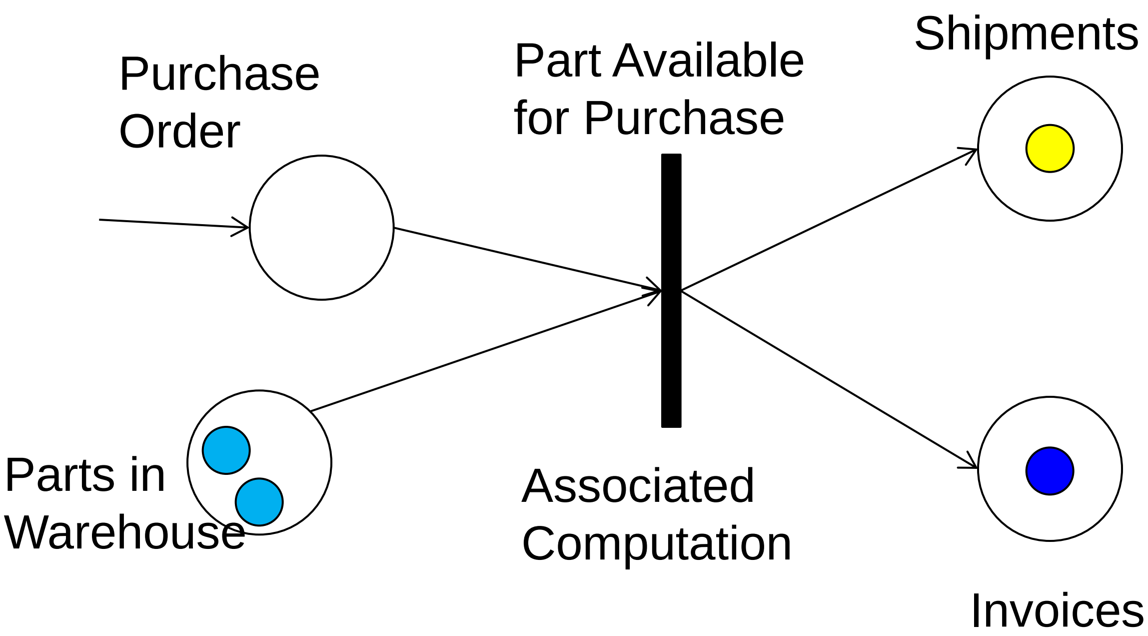


Colored Petri Nets
Simulation step

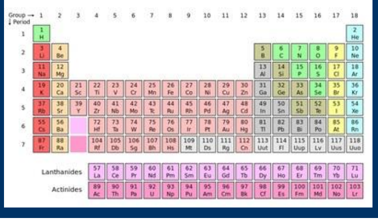


Place: A holder of zero or more (colored) tokens
Token: Marker in place representing available event with data
Outgoing Arc: Connection from an (input) Place to Transition
Incoming Arc: Connection from Transition to (Output) Place
Transition: An intermediary between places
that consumes tokens from input places synchronously
and generates colored tokens in output places
Computes new data values for new tokens using business actions

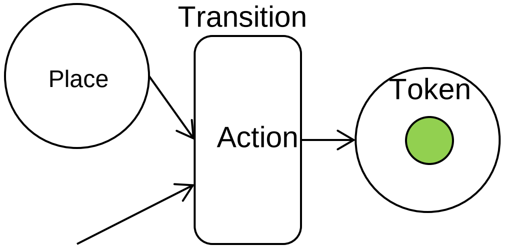


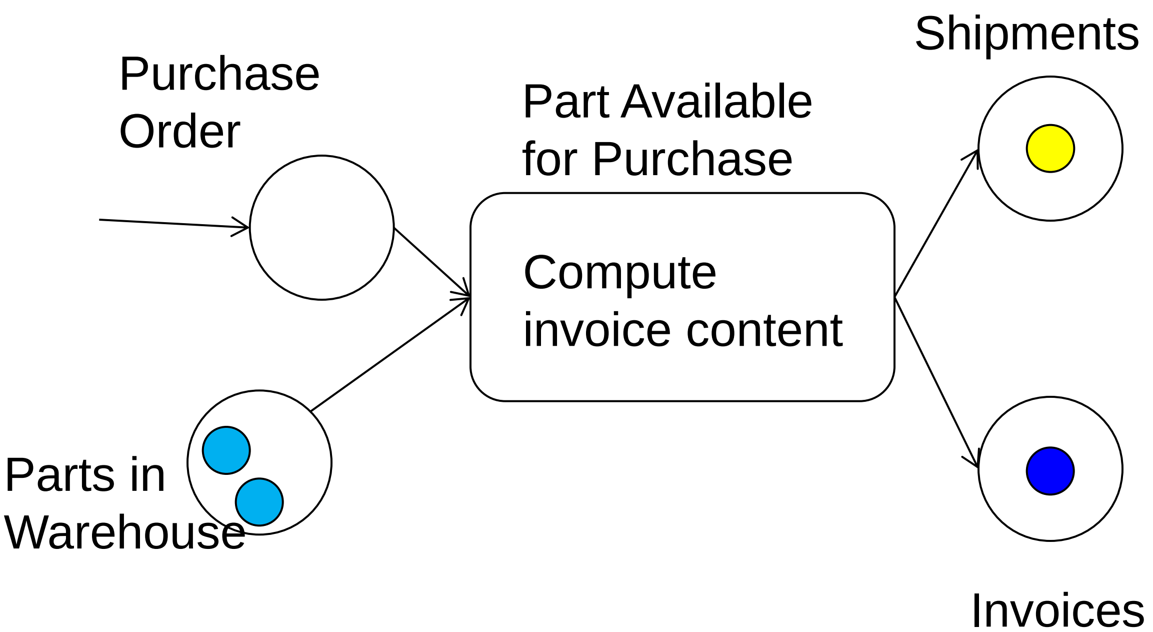


Colored Petri Nets
Slight Notation Change



Place: A holder of zero or more (colored) tokens
Token: Marker in place representing available event with data
Outgoing Arc: Connection from an (input) Place to Transition
Incoming Arc: Connection from Transition to (Output) Place
Transition: An intermediary between places
that consumes tokens from input places synchronously
and generates colored tokens in output places
Computes new data values for new colored tokens (business actions)





This style of computation is called *data flow*

So… what specific information do you want from Business Rules?

There are many possible uses of BR across an organization.

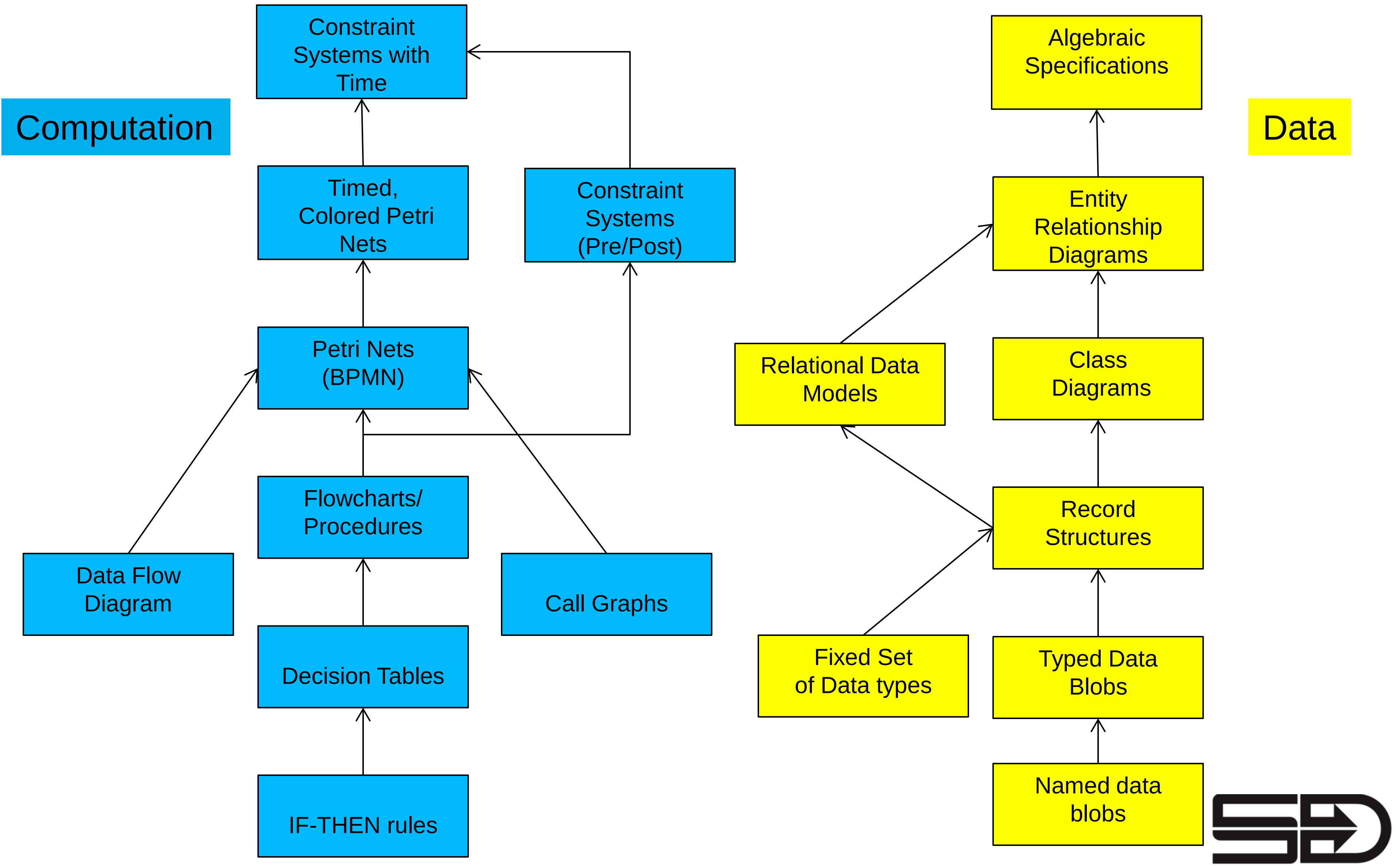
- *Narrowing to one focus enables simpler BR, but limits use.*
- *Widening focus necessitates more complex BR vocabulary*

This creates tension in choice.

So how can you choose?

Consult hierarchy of BR types.

Two hierarchies of program models: code and data



Model and Business Rule Extraction Using Pattern Matching

Model extraction using Pattern Matching

Goal:

Extract models and business rules from legacy systems

Method:

1. ***BA/Programmer identify code idioms representing business actions***
2. ***BA/Programmer define a pattern representing idiom***
3. ***BA/Programmer may define other patterns for same idiom***

4. ***Tool analyze source code to find data flows***
5. ***Tool analyze pattern to find data flows***
6. ***Tool matches pattern to source code using data flows as guide***
7. ***Tool records idiom name as code abstraction***

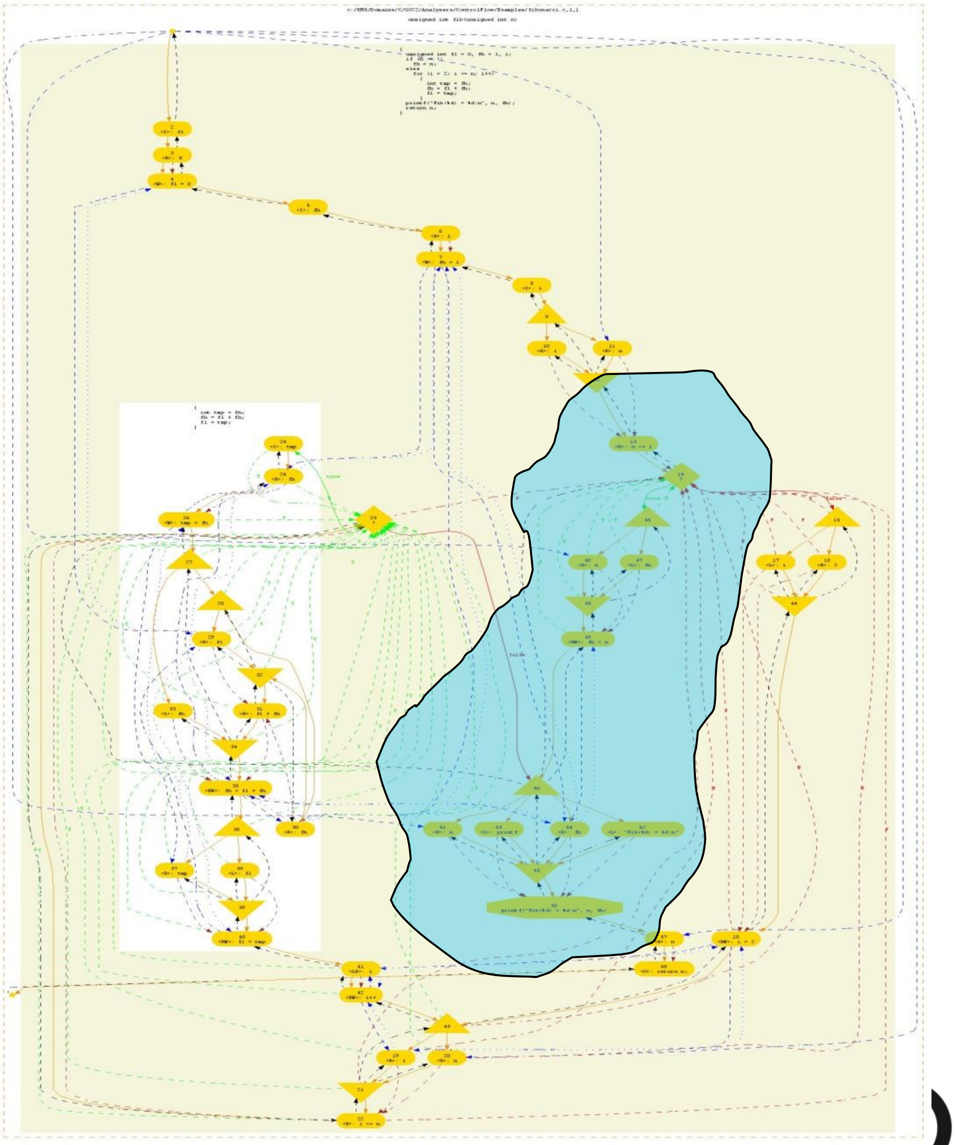
Benefits:

Matched code fragments are instances of business actions
Code variations equivalent to pattern are found

Not discussed: How to build business vocabulary or data models

What's inside a Computer Program?
A Colored Petri Net ("Data flow")

```
int fibonacci(n)
{ unsigned int fl= 0, fh = 1, i;
  if (n <=1 )
     fh = n;
  else
      for (i= 2; i<=n; i++) {
         int tmp = fh;
         fh =fl + fh;
         fl = tmp;
      }
      print ("Fib(%d) = %d\n", n, fh);
      return n;
}
```



Big example wouldn't fit on football field…

Insight:
Maybe we can abstract away this detail

accumulate(fib#s)

(Analysis): Data Flow patterns:

Matching code with dataflows, *not syntax*

```
int getBankCode(int bn) {
        int bc;

        if (bn > 10 & bn <= 25)
                bc = 3;
        else
                bc = 0;

        return bc;
}
```

a "bank classification" idiom
Representing a business computation

```
default base domain C~ISO9899c1990.

public data flow pattern
   classify_bank(bank_number:IDENTIFIER<~,
        bank_code:IDENTIFIER~>):statement_seq
  = "if (\bank_number > 10 & \bank_number <= 25)
       \bank_code = 3; // bank of ethel
     else
       \bank_code = 0; // unknown bank number
    ".
```

Data flow pattern for idiom

```
int displayInfo(int rn) {
        boe_lower = 10;
        bank_number = lookup(rn);
        tmp = bank_number;
        boe_upper = 25;
        chkh = tmp <= boe_upper;
        if (!chkh)
                logOutOfRange(tmp);
        chkl = tmp > boe_lower;
        if (!chkl)
                logOutOfRange(tmp);
        chkr = chkh & chkl;
        if (chkr)
                logWithinRange(tmp);
        if (chkr) {
                boe_code = 3;
                printf("Bank of Ethel\n");
                tmp = boe_code;
        } else {
                u_code = 0;
                printf("Error: Unkown bank number.\n");
                tmp = u_code;
        }
        displayRecord(tmp,rn);
        return tmp;
}
```

Data flow match of idiom woven into code

COBOL tax computation Patterns

```
COMPUTE-TOTAL.
  MULTIPLY QUANTITY BY PRICE GIVING TOTAL-AMOUNT.
  IF TOTAL-AMOUNT > DISCOUNT-THRESHOLD
    MULTIPLY TOTAL-AMOUNT BY DISCOUNT-PERCENT
      GIVING DISCOUNT-AMOUNT
    DIVIDE 100 INTO  DISCOUNT-AMOUNT
    SUBTRACT DISCOUNT-AMOUNT FROM TOTAL-AMOUNT.
  ADD ONE TO VAT-RATE GIVING TAX-ADJUSTMENT.
  MULTIPLY TAX-ADJUSTMENT INTO TOTAL-AMOUNT.
  DISPLAY COMPANY-NAME.
  DISPLAY "Total: ", TOTAL-AMOUNT.
```

```
define TaxStyle = { `Added`, 'Multiplied' }
```

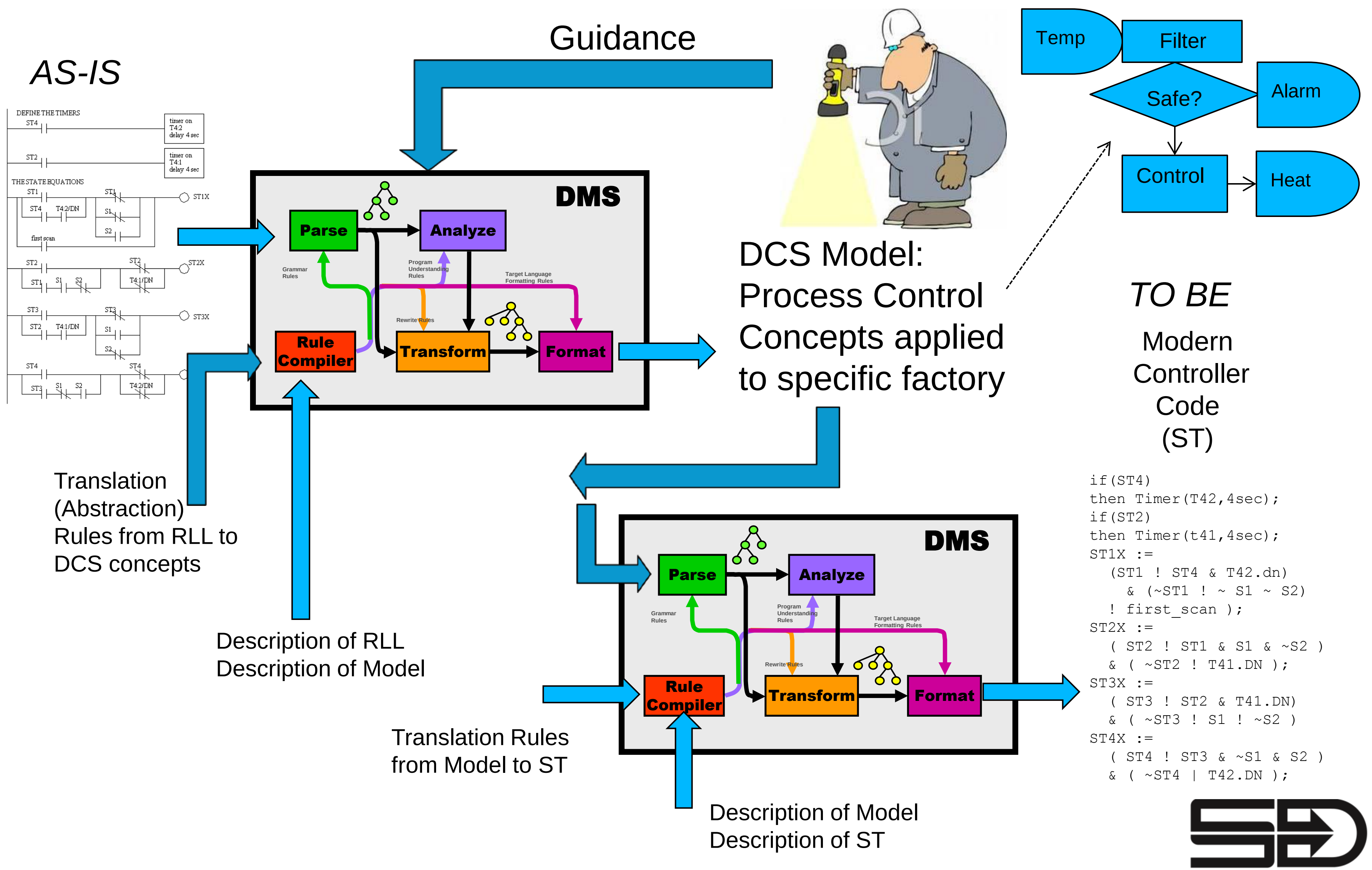
```
data flow pattern ComputeTax_by_multiplying(TaxRate:Constant,
                              Total:IDENTIFIER)
     :StatementSequence
  "Compute \Total = 1.0 + \TaxRate"
   if Value(TaxRate)>0.0 and Value(TaxRate)<1.0;
```

```
COMPUTE-INVOICE.
  MULTIPLY AMOUNT BY VAT-RATE GIVING TAX.
  Compute INSURANCE = INSURANCE_RATE * AMOUNT.
  ADD TAX TO AMOUNT.
  ADD INSURANCE TO AMOUNT GIVING INVOICE_TOTAL.
```

```
data flow pattern ComputeTax_by_adding(TaxRate:Constant,
                                   Total:IDENTIFIER)
           :StatementSequence
   Temp:IDENTIFIER
  "MULTIPLY \Total BY \TaxRate GIVING \Temp.
   ADD \Temp TO \Total"
   if Value(TaxRate)>0.0 and Value(TaxRate)<1.0
```

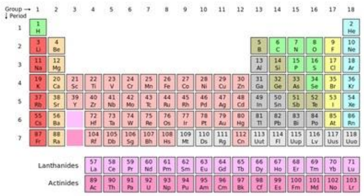
```
data flow pattern ComputeTax(TaxRate:Constant,
                             Total:IDENTIFIER):
               <HowTaxed:  TaxStyle>:
               StatementSequence
  case HowTaxed
     when `Added`
     ComputeTax_by_adding(TaxRate,Total)
     when 'Multiplied'
     ComputeTax_by_multiplying(TaxRate,Total)
  esac;
```

Model Based Migration (Dow Chemical)



Summary

Wide variety of "business rule" schemes
are unified by Colored Petri Net models
augmented by formal data models



Two major components:

Data and business rule computation models
Process/ data flow with synchronization

Differs from BPMN:

How it models data transfers as part of synchronization
How it compute business actions based on formal data model

Key to capturing business rules from code

By recognizing dataflow idioms as business abstractions
Major success in migrating must-not-fail factory
Technology at early-adopter stage

Automated Extraction of Business Rules and Models from Code

Abstract

Everybody talks about "business rules", and how to extract them from code. The definitions vary wildly, and the procedure to extract them are largely informal. This confuses everybody about the nature of business rules and what exactly happens as they are extracted.

This talk will be a synthesis of ideas from the program analysis, reverse engineering, model extraction and business rules extraction community. We will discuss the concept of abstraction as a unifying principle that ties these ideas together in a coherent framework, showing how decision tables, BPEL-style notations, models and domain-specific languages are all variations on a theme.

We will discuss the kind of technology that is required to enable the analysis of code by tracing information flows from system inputs through code to system outputs, and reverse engineering from code idioms with interactive guidance of the process by a business rule analyst. As a case study, we will discuss how we were able to extract reliable models of a factory control process from extremely low level code for Dow Chemical industrial plant controllers using pattern matching technology to recognize common code idioms and design choices. Finally, we will discuss how this technology is likely to evolve for use in the broader business rule cor

Learning Objective 1 *	Learning Objective 2 *	Learning Objective 3 *
What are abstractions, and how are they related to business rules.	Show how computer code implements abstractions	Show how information flow and pattern matching can be used to support extraction of business rules and abstraction

Abstract

Presentation Level	
Advanced	
Kind of Presentation	
1-hour presentation	
Select a General Focus	
? The Evolving Analysis and Design Landscape ? Leveraging Technology ? Fast Forward	
Select a Special Focus	
? Business Rules and Decisions	
Additional focus or positioning of content	
Keyword 1	Keyword 2
abstraction	information flows
design recovery	automation
pattern matching	
How do you plan on conducting the session?	
Traditional presentation	

Speaker Biography

Ira Baxter, Ph.D., has been building system software since 1969. After founding a microprocessor software house in the 1970s, he returned to graduate school at UC Irvine to study reuse of knowledge supporting software maintenance and evolution. On completing his Ph.D. in 1990, he joined Schlumberger as research scientist automating the generation of supercomputer programs for oil field exploration. In 1995, he founded Semantic Designs, where he has been architect/implementer of the Design Maintenance System(R), providing automated program analysis and transformation to large-scale legacy systems.

He has been project lead on a variety of massive code migration and re-architecting projects, including work with Dow Chemical to automate the extraction of models from factory control code. Dr. Baxter has been active in Software Engineering and Maintenance and other conferences since 1983, including co-chairing of the International Conference on Software Maintenance.

Thank You

Contact

Randal Matthias

Business Development Manager

Rmatthias@semanticdesigns.com

1-512-250-1018 x172

www.SemanticDesigns.com