This Week

Monday

- Solving Decision Problems Using VBA (Part II)

Wednesday

- Lab Exercise: Service Parts Optimization

Topics

- Swifty Service Parts
  - The Nature of Service Agreements
  - Modeling the Problem
  - Solving the Problem
- Working with VBA Ranges

A Problem Solving Framework

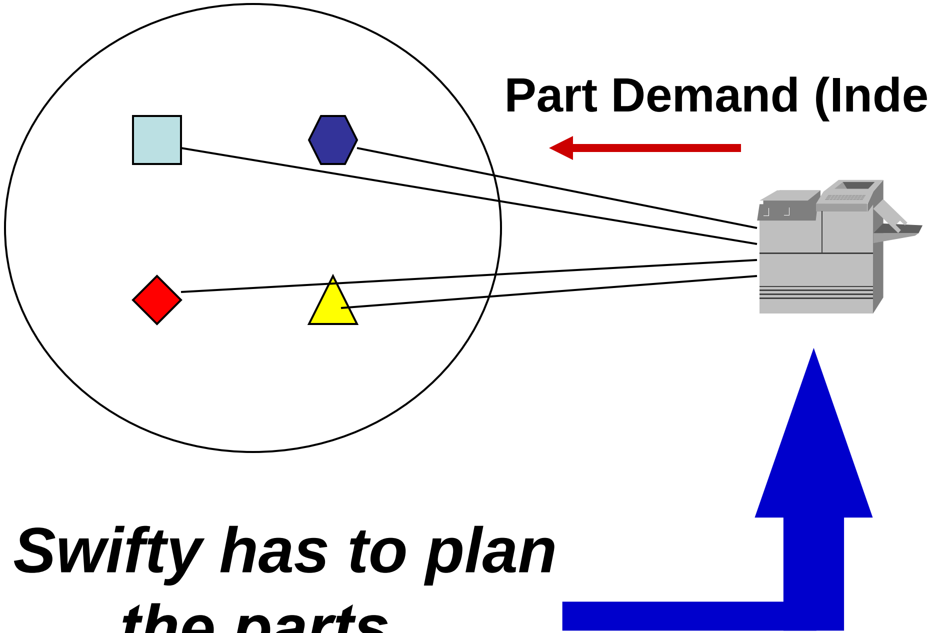
1. *Define the Problem*
2. *Collect and Organize Data*
3. *Characterize Uncertainty and Data Relationships*
4. *Build an Evaluation Model*
5. *Formulate a Solution Approach*
6. *Evaluate Potential Solutions*
7. *Recommend a Course of Action*

Swifty Service Parts

- ***The Nature of Service Agreements***
- **Modeling the Problem**
- **Solving the Problem**

The Nature of Service Agreements

1. **Service agreements focus on *products*, not service parts.**

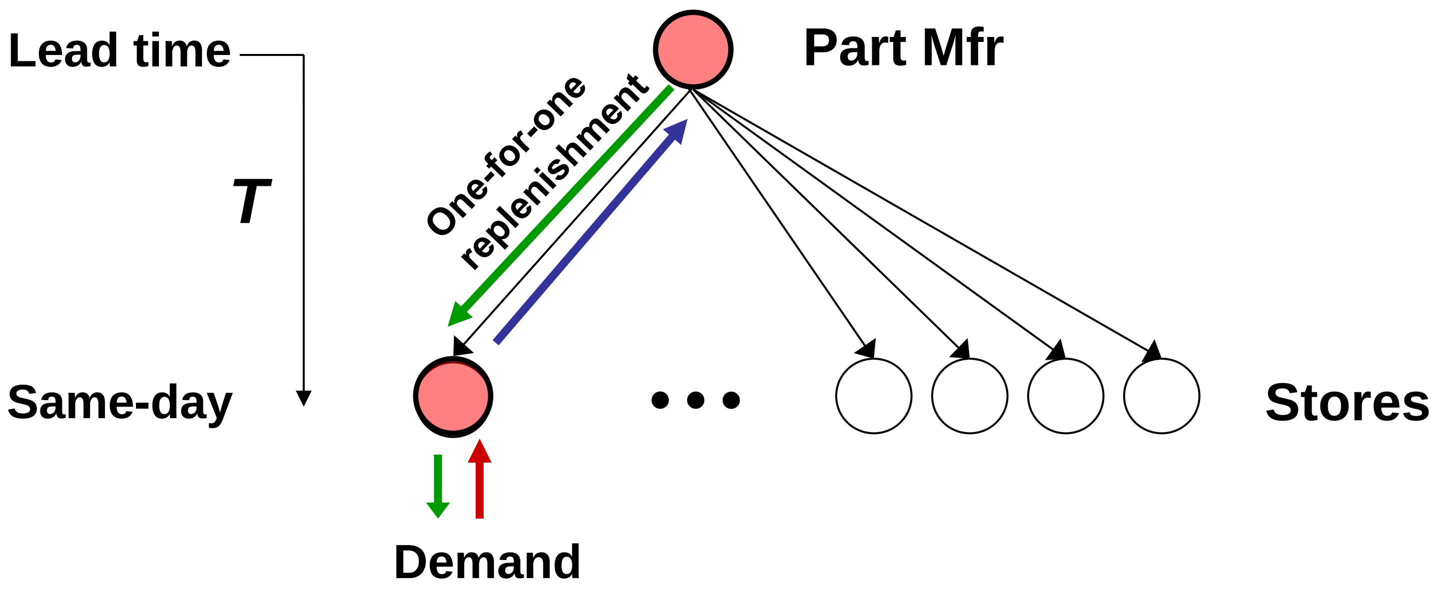


… to keep the customers' products in service.

Swifty has to plan the parts…

The Nature of Service Agreements

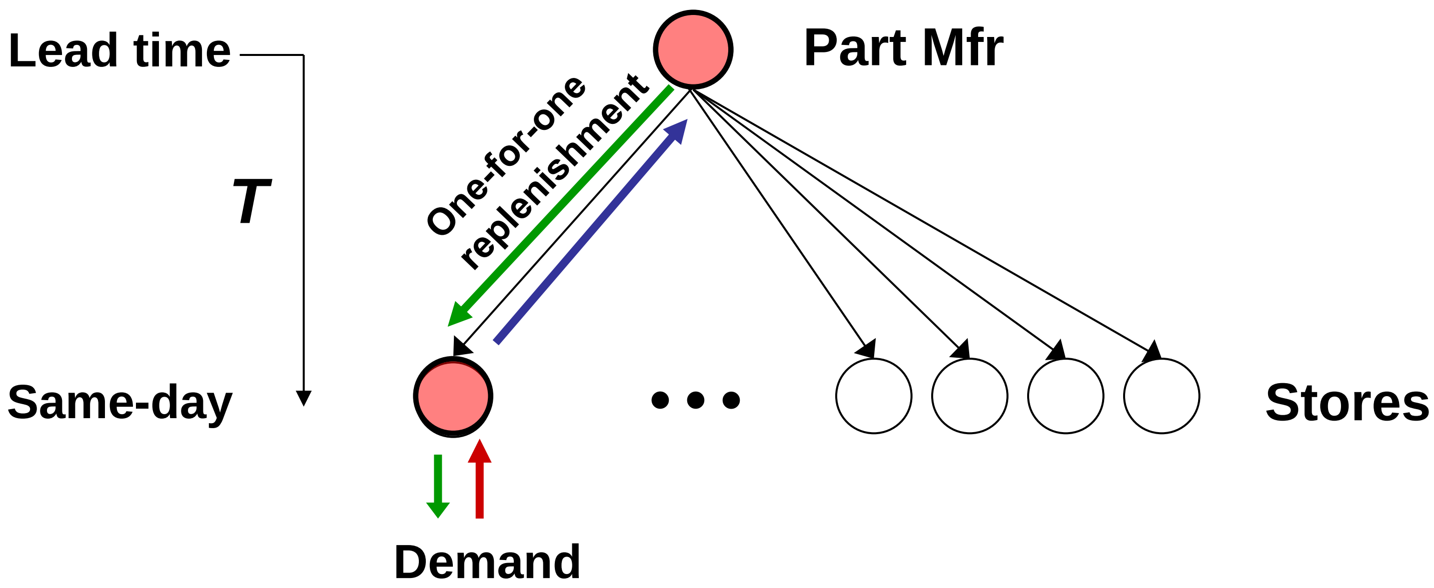
2. Service level guarantees are ***time-based***, and replenishment times are driven by the ***distribution network structure***.



In Swifty's case, the service level guarantees of concern are same-day (i.e., immediate) service guarantees at the stores.

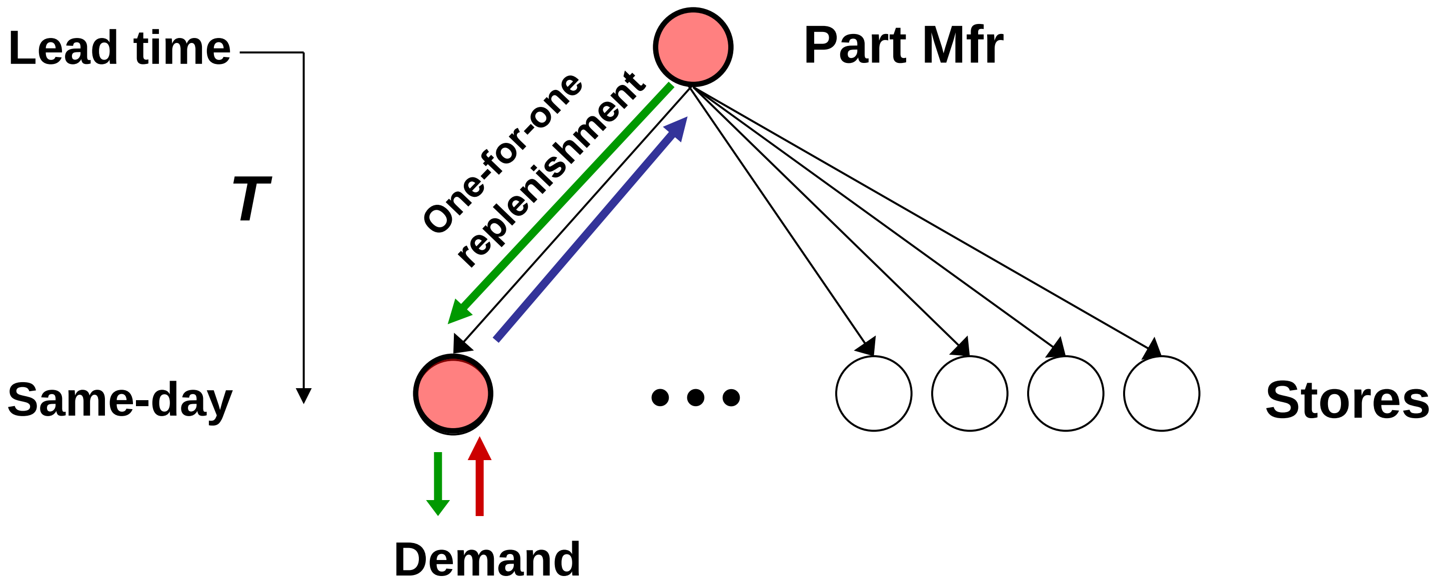
The Nature of Service Agreements

3. Service level guarantees are driven by ***impact of the product on the customer's business*** (or perception of impact).



For each product class at each store, Swifty needs to achieve a ***95% same-day customer service level***.

How Much Inventory at Each Store?



Challenges:

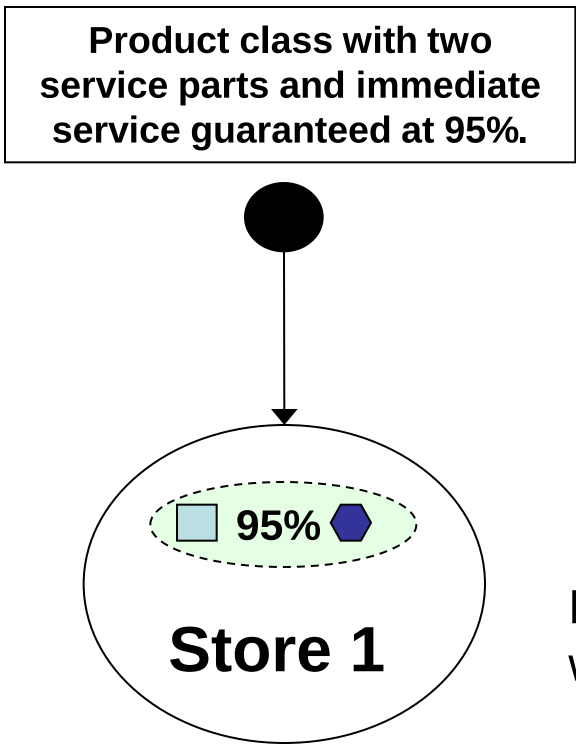
- ***Write down*** **the problem**
- ***Evaluate*** **the key components**
- ***Solve*** **it (efficiently)**

Swifty Service Parts

- **The Nature of Service Agreements**
- ***Modeling the Problem***
- **Solving the Problem**

Modeling Service Level Constraints

- **Each product class demanded at a store gives rise to a *service level constraint*.**
- **Each service level constraint can be written as the sum of weighted *fill rates*.**



Immediate fill rate for part $\square$ at store 1.

$$\frac{\lambda_{\square}}{\lambda_{\square,\hexagon}} f_{\square 1} + \frac{\lambda_{\hexagon}}{\lambda_{\square,\hexagon}} f_{\hexagon 1} \geq 0.95$$

Likelihood that product service will require item $\square$ at store 1.

Problem Formulation

Choose base stock levels s_{ij} to:

$$\text{minimize} \boxed{\left(\sum_{\text{items } i} \sum_{\text{Stores } j} c_i \cdot s_{ij} \right)}$$

System Inventory Investment

$$\text{subject to:} \sum_{\text{items } i} \boxed{w_{ij}} \cdot \boxed{f_{ij}(s_{ij})} \geq \boxed{F_j} \quad \forall \text{ stores } j$$

Service level guarantee at store *j*

$$s_{ij} \geq 0 \text{ and integer } \forall i,j$$

Relative likelihood that service part *i* will be demanded at store *j*

Immediate fill rate of service part *i* at store *j* (depends upon the base stock level of part *i* at location *j*)

Swifty Service Parts

- **The Nature of Service Agreements**
- **Modeling the Problem**
- ***Solving the Problem***

Simplifying Facts

$$\textbf{minimize} \quad \left(\sum_{\text{items } i} \sum_{\text{Stores } j} c_i \cdot s_{ij} \right)$$

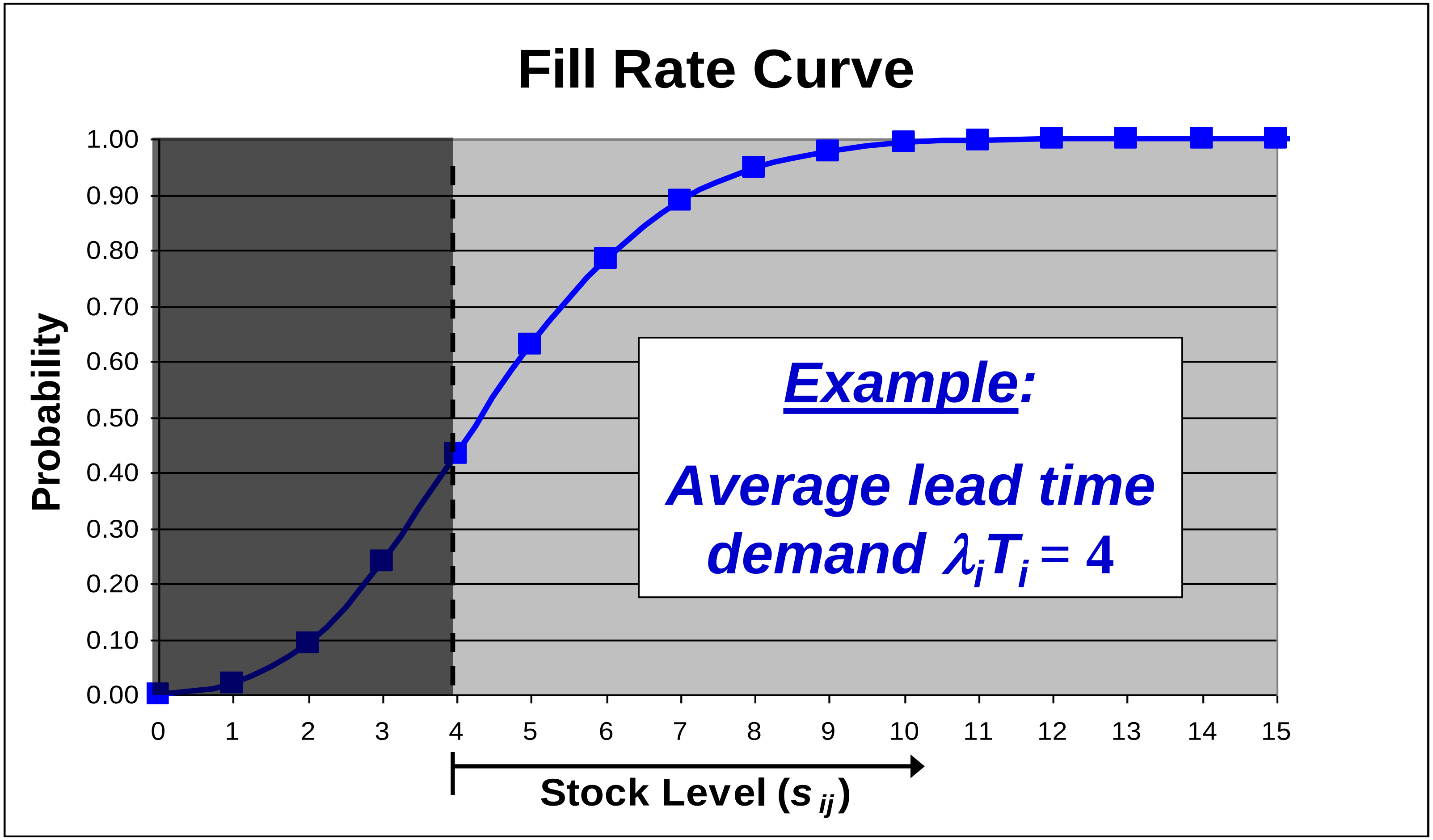
$$\textbf{subject to:} \sum_{\text{items } i} w_{ij} \cdot f_{ij}(s_{ij}) \geq F_j \quad \forall \textbf{ stores } j$$

$$s_{ij} \geq 0 \text{ and integer } \forall i,j$$

- **The problem is separable by store.**
- **Because the part demand processes are independent and Poisson, the base stock replenishment policy implies that:**

$$f_{ij}(s_{ij}) = \Pr[X_{ij} < s_{ij}], \textbf{ where } X_{ij} \sim Po(\lambda_{ij} T_{ij})$$

One More Simplifying Fact



For $s_{i1} \geq \lfloor \lambda_{i1} T_{i1} \rfloor$, the part fill rate function is *concave*.

Single Store Problem

<u>For Store *j* = 1:</u>

$$\textbf{minimize} \quad \sum_{\textbf{items } i} c_i \cdot s_{i1}$$

$$\textbf{subject to:} \quad \sum_{\textbf{items } i} w_{i1} \cdot f_{i1}(s_{i1}) \;\geq\; F_1$$

If we replace $\cancel{s_{i1} \geq 0 \ \textbf{and integer}} \quad \forall i$

with $s_{i1} \geq \lfloor \lambda_{i1} T_{i1} \rfloor$ and integer

then all part fill rate functions are <u>concave over the relevant range</u>, and the problem can be solved to near-optimality using <u>simple greedy marginal analysis</u>.

Marginal Analysis Algorithm

For store *j* = 1:

(1) For every part type i, set $s_{i1} = \lfloor \lambda_{i1} T_{i1} \rfloor$

(2) While $\sum_{\text{items } i} w_{i1} \cdot f_{i1}(s_{i1}) < 95\%$:

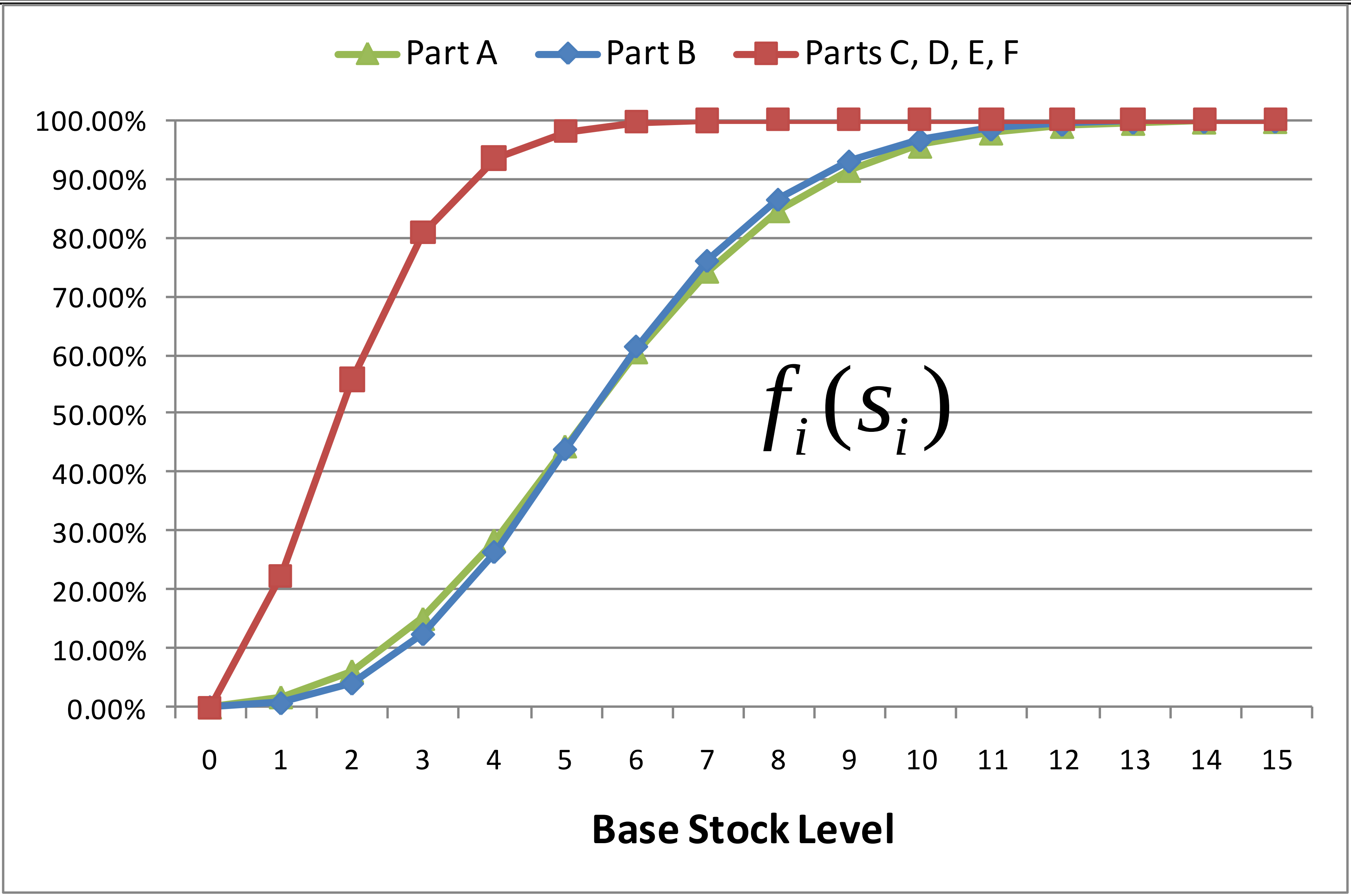
(a) For every part type i, compute:

$$\Delta f_{i1}(s_{i1}+1) = f_{i1}(s_{i1}+1) - f_{i1}(s_{i1})$$

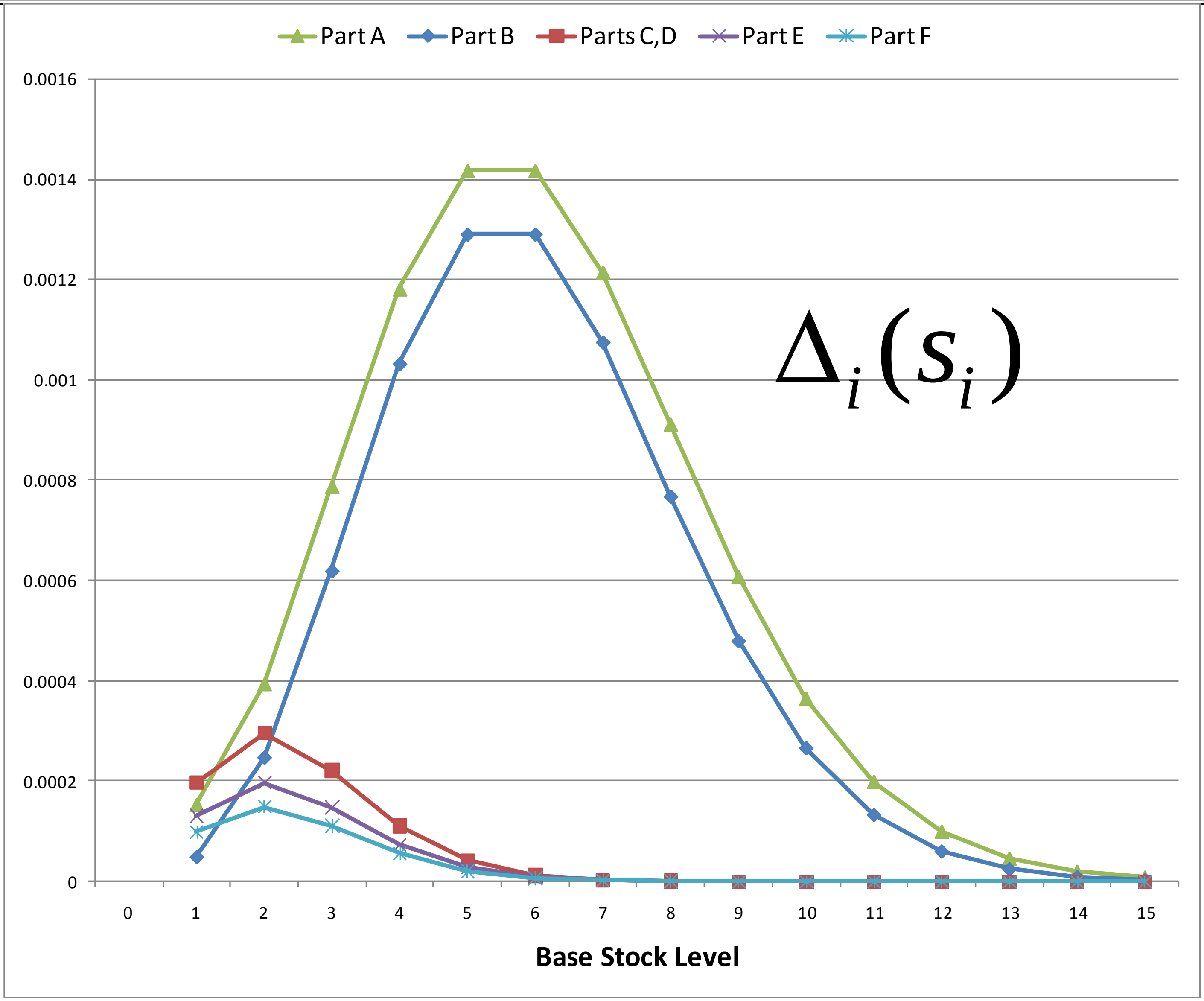
(b) Increment s_{i1} for the part type i with maximum Δ_{i1}, where:

$$\Delta_{i1} = \frac{w_{i1} \cdot \Delta f_{i1}(s_{i1}+1)}{c_i}$$

Plotter Part Fill Rate Curves



Marginal Change in CSL per $ Spent



Working With VBA Arrays

- Static Arrays
- Dynamic Arrays
- Option Base Statement

Static Arrays

```
Dim part_name As String
Dim counter As Integer
Dim SaveStockArray(1 To 6) As Integer

For counter = 1 To 6
    part_name = Range("Part_List").Cells(1,counter).Value
    SaveStockArray(counter) = Range(part _name & "_stock").Value
Next counter
```

*Uses a **For** loop to run through the 6 parts and set the array entries one by one*

- The code above assumes that you ***know*** there are 6 parts ahead of time. It also implicitly assumes that the number of parts will never change. In general, this is not good coding practice.
- Static arrays are useful for holding values associated with a known number of entities that are not likely to change (e.g., 12 months).

Dynamic Arrays

```
Dim part_name As String
Dim counter As Integer
Dim SaveStockArray() As Integer
Dim num_parts As Integer
Dim cell As Range

    num_parts = Range("Part_List").Cells.Count

    ReDim SaveStockArray(1 To num_parts)

    counter = 1
    For Each cell In Range("Part_List")
        part_name = cell.Value
        SaveStockArray(counter) = Range(part _name & "_stock").Value
        counter = counter + 1
    Next cell
```

*Uses the **Cells** and **Count** properties to get the number of parts, and then uses it to size SaveStockArray*

*Uses **ReDim** to size the array*

*Uses a **For Each** loop to run through the part list and set the array entries one by one*

Option Base Statement

```
Option Base 1
```

*Specifies that, **as a default**, all arrays will begin with index 1. If omitted, array indices begin with 0 (zero) as a default.*

```
Sub SaveStockValues()

    Dim FirstArray(6) As Integer
    Dim SecondArray(1 To 6) As Integer
    Dim ThirdArray(0 To 5) As Integer
    .
    .
    .
End Sub
```

- `Dim FirstArray(6) As Integer` — *FirstArray will begin with index 1 since Option Base 1 is specified. Else, it would begin with 0.*
- `Dim SecondArray(1 To 6) As Integer` — *SecondArray will have indices 1 to 6, <u>regardless</u> of whether Option Base 1 is specified.*
- `Dim ThirdArray(0 To 5) As Integer` — *ThirdArray will have indices 0 to 5, <u>regardless</u> of whether Option Base 1 is specified.*

- In general, it is good coding practice to specify the beginning and ending array indices in the Dim statement, so there is no ambiguity.