# Case Study V: A Help-Desk Service

Prof. Daniel A. Menascé
Department of Computer Science
George Mason University
www.cs.gmu.edu/faculty/menasce.html

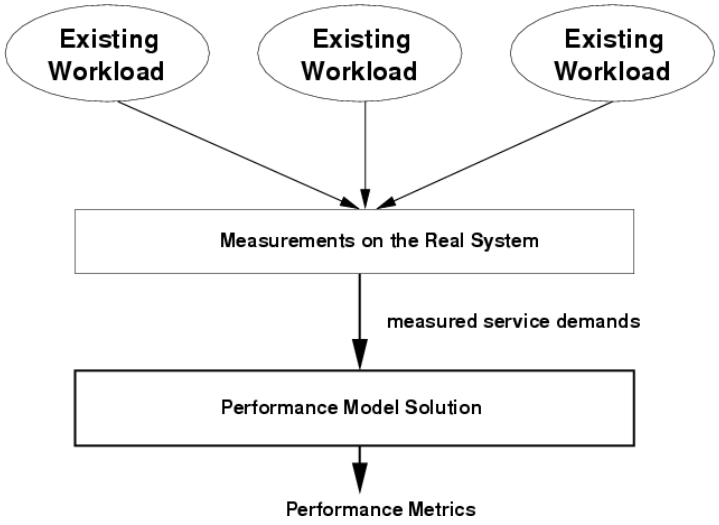




# Copyright Notice

- Most of the figures in this set of slides come from the book "Performance by Design: computer capacity planning by example," by Menascé, Almeida, and Dowdy, Prentice Hall, 2004. It is strictly forbidden to copy, post on a Web site, or distribute electronically, in part or entirely, any of the slides in this file.

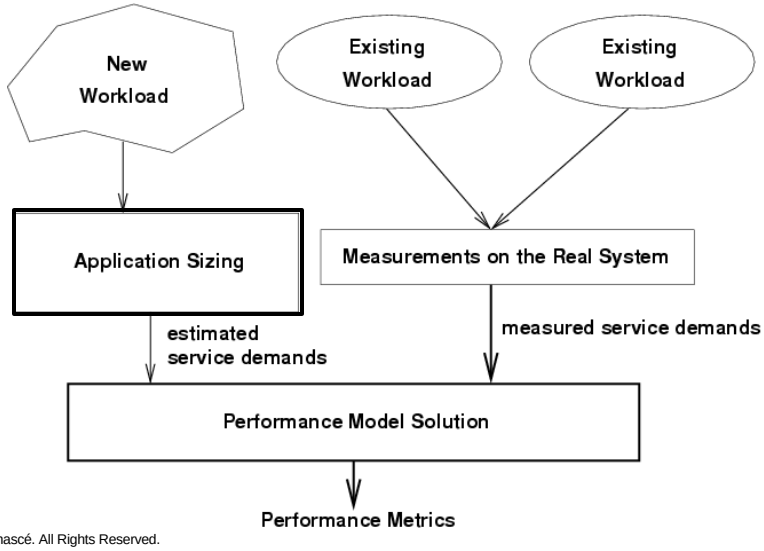
# Capacity Planning for Existing Workloads




3



# Capacity Planning for New Workloads




4

# Software Performance Engineering (SPE)

- A set of methods and techniques to design software systems in such a way that they will adhere to performance requirements once implemented.
- SPE has to be applied at different stages during the life cycle of a software system so that estimates on services demands are continuously refined.
- Software Performance Engineers need to provide feedback to developers during the software development stage to improve the performance of the system in question.

5

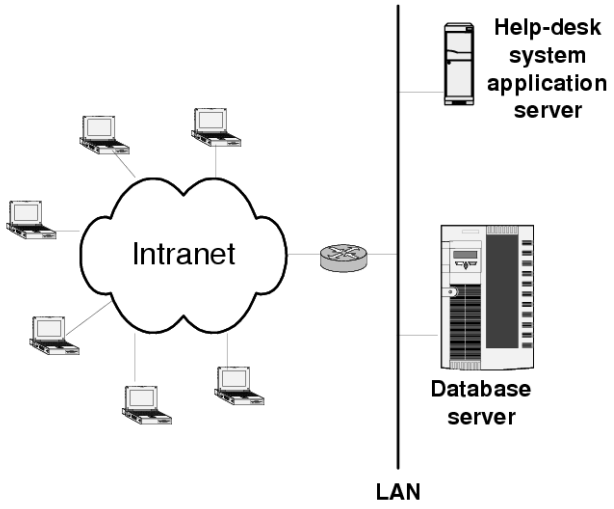


# The Help-Desk System

- A high-tech company has 21,600 employees with access to a variety of computing resources.
- A new help-desk application is being designed:
  - Access to a DB of Frequently Asked Questions (FAQ). Keyword-based search.
  - Creation of help tickets. If the FAQ DB does not solve the problem a help ticket is created.
  - Tracking and verification of the status of open help tickets.

6

# The Help-Desk System




7



# Data for Application Sizing

| | |
|---|---|
| Number of employees | 21,600 |
| Percent employees that access FAQ DB per day | 0.20 |
| Percent access to FAQ DB at peak period | 0.80 |
| Avg. number of queries to FAQ DB per access to FAQ DB | 4.00 |
| Percent of accesses to the FAQ DB that generate ticket creation | 0.35 |
| Percent of employees that create new tickets/day without going to the FAQ database | 0.15 |
| Percent of new ticket creation at peak period | 0.54 |
| Percent of employees that inquire about new ticket status/day | 0.10 |
| Percent of status inquiries at the peak period | 0.90 |
| Peak period duration (in seconds) | 7,200 |

*Arrival rates during peak period (requests/sec)*

| | |
|---|---|
| FAQ queries | 1.9200 |
| New ticket creation | 0.4110 |
| Ticket status | 0.2700 |

| | |
|---|---|
| Period (in days) during which tickets are kept | 365 |

8

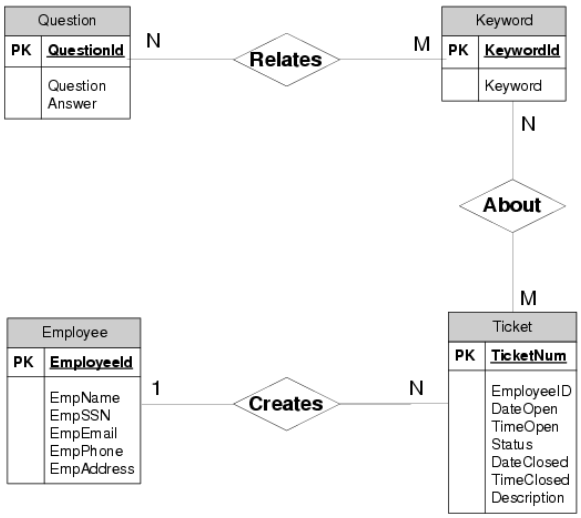
# Workload Intensities

$$\lambda_{\text{FAQ}} = \frac{N_e \times f_{\text{FAQ}} \times P_{\text{FAQ}} \times N_q}{\mathcal{T}}$$
$$= \frac{21,600 \times 0.2 \times 0.8 \times 4}{7,200} = 1.92 \ \text{requests/sec}$$
$$\lambda_{\text{ticket}} = \frac{N_e \times [(f_{\text{FAQ}} \times P_{\text{FAQ}} \times t_{\text{FAQ}}) + (f_t \times P_t)]}{\mathcal{T}}$$
$$= \frac{21,600 \times [(0.2 \times 0.8 \times 0.35) + (0.15 \times 0.54)]}{7,200} = 0.41 \ \text{requests/sec}$$
$$\lambda_{\text{status}} = \frac{N_e \times f_s \times P_s}{\mathcal{T}} = \frac{21,600 \times 0.1 \times 0.9}{7,200} = 0.27 \ \text{requests/sec} \quad (9.2.1)$$

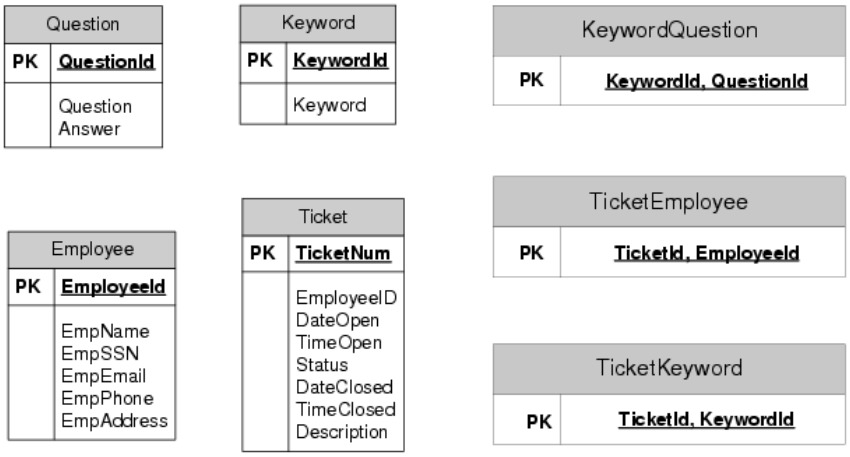




# Entity-Relationship Model for the Database Design



# Database Design for the Help-Desk System








# Database Table Data

| Database Table | Cardinality | Row Size(bytes) | Total size(Kbytes) |
|---|---|---|---|
| Question | 10,000 | 1000 | 10,000 |
| Keyword | 500 | 50 | 25 |
| Employee | 21,600 | 140 | 3,024 |
| Ticket | 1,734,480 | 260 | 450,965 |
| KeywordQuestion | 100,000 | 20 | 2,000 |
| TicketEmployee | 1,734,480 | 20 | 34,690 |
| TicketKeyword | 8,672,400 | 20 | 173,448 |

Assumptions:
- A ticket is kept in the DB for one year.
- One question is associated with 10 keywords, on average.
- One ticket has 5 keywords associated to it, on average.
- Questions are 1000 bytes long.
- Keywords are 50 bytes long.

# Specifying Transaction Logic for SPE Purposes

- Use a language that captures the major structural components of a transaction.
  - Loops and average number of times executed.
  - Branch statements and branching probabilities.
  - Switch statements and case probabilities.
  - Database access (i.e., select and update) statements.
- Estimate number of I/Os per transaction and estimated the CPU time from the number of I/Os using benchmark data.
- Example of such a language: *Clisspe*

13



# Transaction Logic for Query on FAQ Database

```
01 loop #KeywordsPerQuery
02   ! obtain keyword id for a given a keyword
03   select from Keyword where Keyword;
04   ! obtain all question ids for a given keyword id
05   select from KeywordQuestion where KeywordId;
06   ! access the Question table to retrieve selected questions
07   loop #QuestionsPerKeyword
08     select from Question where QuestionId;
09   end_loop;
10 end_loop;
```

14

# Transaction Logic for the Creation of a New Ticket

```
01 ! Access the employee table to the verify existence of an employee
02 select from Employee where EmployeeId;
03 if #ProbValidEmployee
04 then ! create ticket record
05       update Ticket num_rows= 1;
06       ! create a record in the TicketEmployee table
07       update TicketEmployee num_rows= 1;
08       ! create records in the TicketKeyword table
09       update TicketKeyword num_rows= #KeywordsPerTicket;
10 end_if;
```

15



# Transaction Logic for Viewing the Status of Open Tickets

```
01 ! Get all ticket ids for the employee
02 select from TicketEmployee where EmployeeId;
03 ! Retrieve all open tickets for the employee
04 loop #TicketsPerEmployee
05   select from Ticket where TicketId, Status
06 end_loop;
```

16

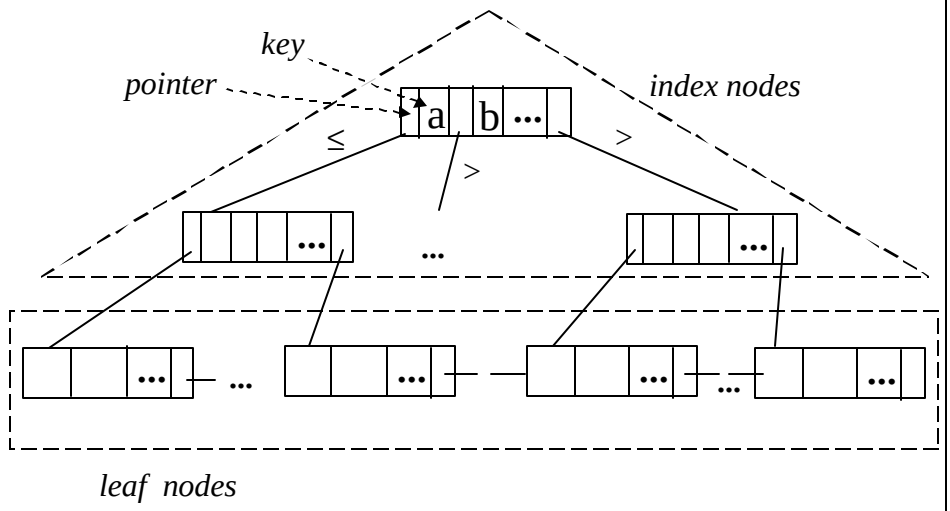
# Estimating Number of I/Os

- Estimate the number of I/Os per database access statement.
  - Consider the existence and types of indexes on the tables.
  - Estimate the number of index blocks accessed.
  - Estimate the number of data pages accessed.





# B*-Tree Indexes



# B-Tree Calculations

*Height of the B-tree* — *Number of entries in the B-tree*

$$h = \lceil \log_k L \rceil$$

*Fanout of the B-tree* — *Fill factor: % of useful space of an index page used to store keys and row pointers.*

$$k = \left\lfloor \frac{(\text{PageSize} - \text{HeaderSize}_i) \times f_i}{ks + rp} \right\rfloor$$

*Size (in size) of a key in the index* — *Size (in size) of a row pointer*

19



# Indexes for the Database Tables

| | |
|---|---|
| PageSize (bytes) | 2,048 |
| Header size (bytes) | 20 |
| Percent useful space | 0.71 |
| Size (bytes) row pointer | 4 |

***Index Info*:**

| Table | IndexKey | KeySize(bytes) | Fanout | Height |
|---|---|---|---|---|
| Question | QuestionId | 10 | 102 | 2 |
| Keyword | KeywordId | 10 | 102 | 2 |
| Employee | EmployeeId | 10 | 102 | 3 |
| Ticket | TicketId | 10 | 102 | 4 |
| KeywordQuestion | (KeywordId,QuestionID) | 20 | 59 | 3 |
| KeywordQuestion | KeywordId | 10 | 102 | 3 |
| TicketEmployee | (TicketId,EmployeeId) | 20 | 59 | 4 |
| TicketEmployee | EmployeeId | 10 | 102 | 4 |
| TicketKeyword | (TicketId,KeywordId) | 20 | 59 | 4 |

20

# Estimating No. I/Os Due to a Select Statement

- Case 1: No index is available. Do a table scan (all data pages have to be accessed).

$$ndp_T = \left\lceil \frac{\text{NumRows}_T}{nrp_T} \right\rceil$$

*No. rows of T that can be stored in a data page.*

- Case 2: An index is available. The number of I/Os is the height of the B-tree minus 1 (the root is assumed to be in memory) + number of data pages retrieved.

21



# Database Table Sizes

***Data Page Calculations:***

| Database Table | RowSize(bytes) | Number Rows Per Page | Number Data Pages |
|---|---|---|---|
| Question | 1000 | 1 | 10,000 |
| Keyword | 50 | 31 | 17 |
| Employee | 140 | 11 | 1,964 |
| Ticket | 260 | 6 | 289,080 |
| KeywordQuestion | 20 | 73 | 1,370 |
| TicketEmployee | 20 | 73 | 23,760 |
| TicketKeyword | 20 | 73 | 118,800 |

22

# Computing Number of I/Os

| | |
|---|---|
| KeywordsPerQuery | 2 |
| QuestionsPerKeyword | 20 |
| ProbValidEmployee | 0.9 |
| KeywordsPerTicket | 5 |
| TicketsPerEmployee | 80.3 |

***Transaction: FAQ***

| Statement no. | Statement type | Table | Where | Index | No. of execs | Prob. Exec | Index I/Os | Data I/Os | Total I/Os |
|---|---|---|---|---|---|---|---|---|---|
| 3 | select | Keyword | Keyword | none | 2 | 1.0 | 0 | 17 | 34 |
| 5 | select | KeywordQuestion | KeywordId | KeywordId | 2 | 1.0 | 2 | 20 | 44 |
| 8 | select | Question | QuestionId | QuestionId | 40 | 1.0 | 1 | 1 | 80 |
| | | | | | | | | | 158 |

Statement 3: Table scan
2 x 1.0 x 17 = 34

Statement 5: Index-based retrieval
2 x 1.0 x ((3-1)+20) = 44





# Computing Number of I/Os

| | |
|---|---|
| KeywordsPerQuery | 2 |
| QuestionsPerKeyword | 20 |
| ProbValidEmployee | 0.9 |
| KeywordsPerTicket | 5 |
| TicketsPerEmployee | 80.3 |

***Transaction: New Ticket***

| Statement no. | Statement type | Table | Where | Index | No. of execs | Prob. Exec | Index I/Os | Data I/Os | Total I/Os |
|---|---|---|---|---|---|---|---|---|---|
| 2 | select | Employee | EmployeeID | EmployeeId | 1 | 1.0 | 2 | 1 | 3.0 |
| 5 | update | Ticket | | TicketId | 1 | 0.9 | 3 | 1 | 3.6 |
| 7 | update | TicketEmployee | | (TicketId,EmployeeId) | 1 | 0.9 | 3 | 1 | 3.6 |
| | | | | EmployeeId | 1 | 0.9 | 3 | 0 | 2.7 |
| 9 | update | TicketKeyword | | (TicketId,KeywordId) | 1 | 0.9 | 15 | 5 | 18.0 |
| | | | | | | | | | 30.9 |

Statement 2: index-based retrieval
1 x 1.0 x ((3-1)+1) = 3

# Computing Number of I/Os

| | |
|---|---|
| KeywordsPerQuery | 2 |
| QuestionsPerKeyword | 20 |
| ProbValidEmployee | 0.9 |
| KeywordsPerTicket | 5 |
| TicketsPerEmployee | 80.3 |

***Transaction: View Tickets***

| Statement no. | Statement type | Table | Where | Index | No. of execs | Prob. Exec | Index I/Os | Data I/Os | Total I/Os |
|---|---|---|---|---|---|---|---|---|---|
| 2 | select | TicketEmployee | EmployeeId | EmployeeId | 1 | 1.0 | 3 | 80.3 | 83.3 |
| 5 | select | Ticket | TicketId, Status | TicketId | 80.3 | 1.0 | 3 | 1.0 | 321.2 |
| | | | | | | | | | 404.5 |

Statement 5: index-based retrieval

80.3 x 1.0 x ((4-1)+1) = 321.2

A very large number of I/Os!





# Computing Service Demands

***Service Demand Computations:***

| | | |
|---|---|---|
| CPU time per 2Kbyte page | 0.0015 | sec/IO |
| Service time per I/O (sec) | 0.008 | sec/IO |

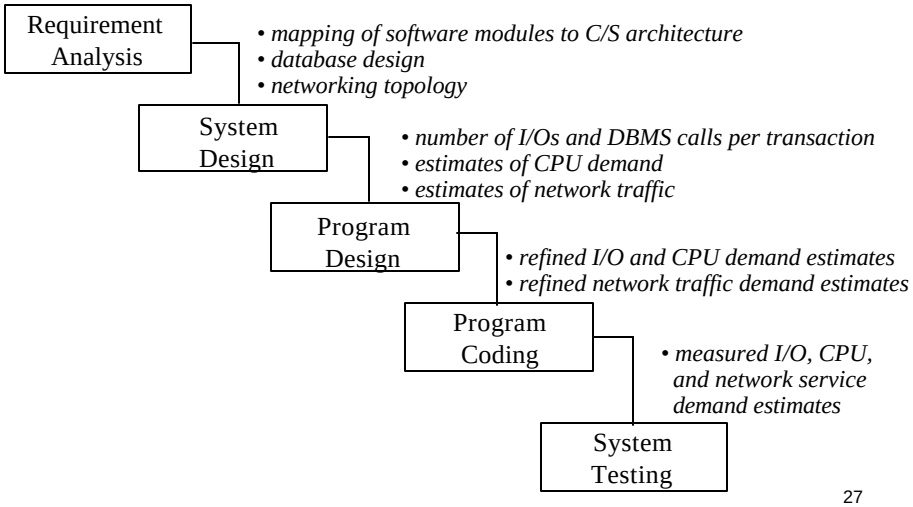
| | FAQ | Create Ticket | View Tickets |
|---|---|---|---|
| CPU | 0.237 | 0.046 | 0.607 |
| Disk | 1.264 | 0.2472 | 3.236 |

Very large service demand.
Disk utilization exceeds 100%.

Solution: instead of storing open and closed tickets in the same table, keep only open tickets in the Tickets table and create another table to archive closed tickets for one year.

## The Software Development Life Cycle and Inputs to SPE

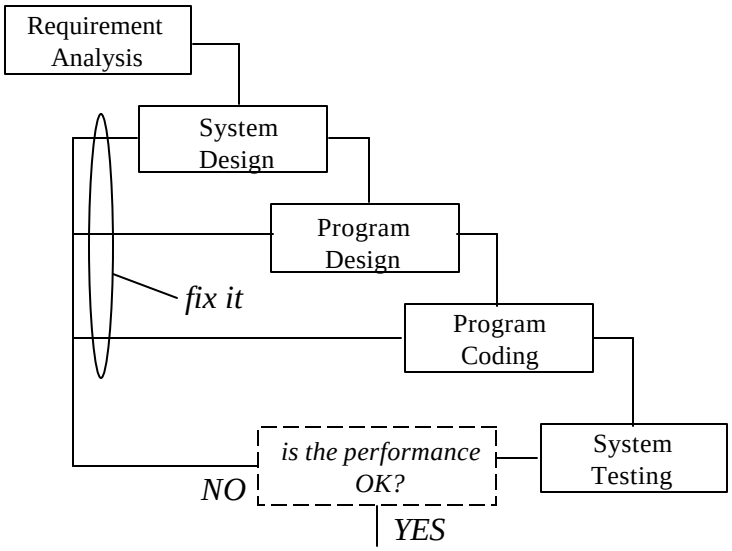
- *service levels (response times, throughputs, etc)*
- *hardware/software base (client and software platforms, networking technologies, DBMSs)*




27



## Traditional Software Development Life Cycle




28

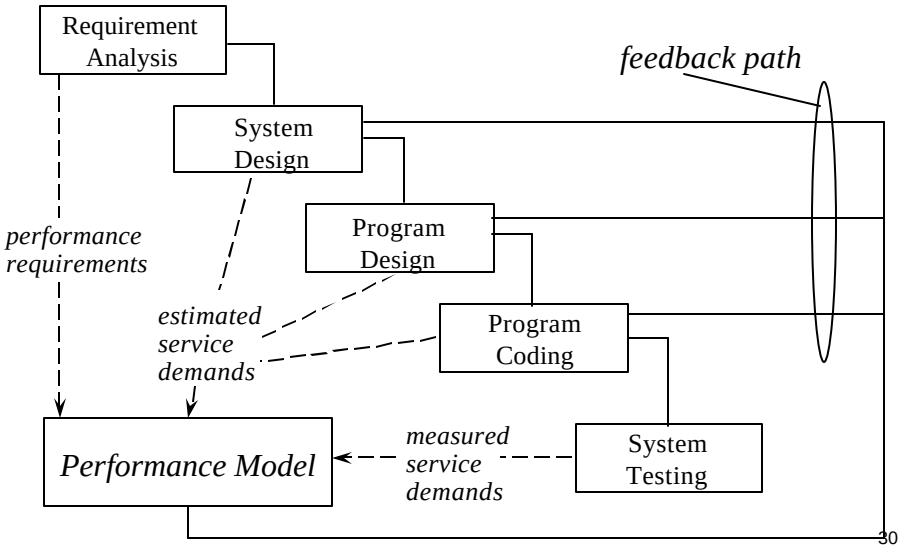
## Traditional Software Development Life Cycle

- Common approach:
  - consider Functional Requirements only during development and check Performance Requirements at the end.
  - fix the system if performance is not good!
- Problem:
  - it is very costly and time consuming to fix the problem after the system is ready!
  - fixing the problem may imply in major software rewrites.

29



## Integrating Software Performance Engineering Into the Software Development Life Cycle




30

# Service Demand Generation

$$D_{i,r} = \sum_{s \in S_{i,r}} n_s \times p_s \times D_{i,r}^s$$

where,

$D_{i,r}^s$: average service demand at device $i$ for class $r$ due to statement $s$.

$S_{i,r}$ : set of statements that generate demands at device $i$ for class $r$.

$n_s$ : average number of time that statement $s$ is executed.

$p_s$ : probability that statement $s$ is executed.

31