LOG 8371E
Software Quality Engineering

Lecture 06:
Software Performance Models

Armstrong Foundjem Ph.D. — Winter 2024

Review:
Software Performance Engineering

(Ultra) Large-Scale Software Systems

stackoverflow

4 million users

2600-3000 req/sec on most weekdays

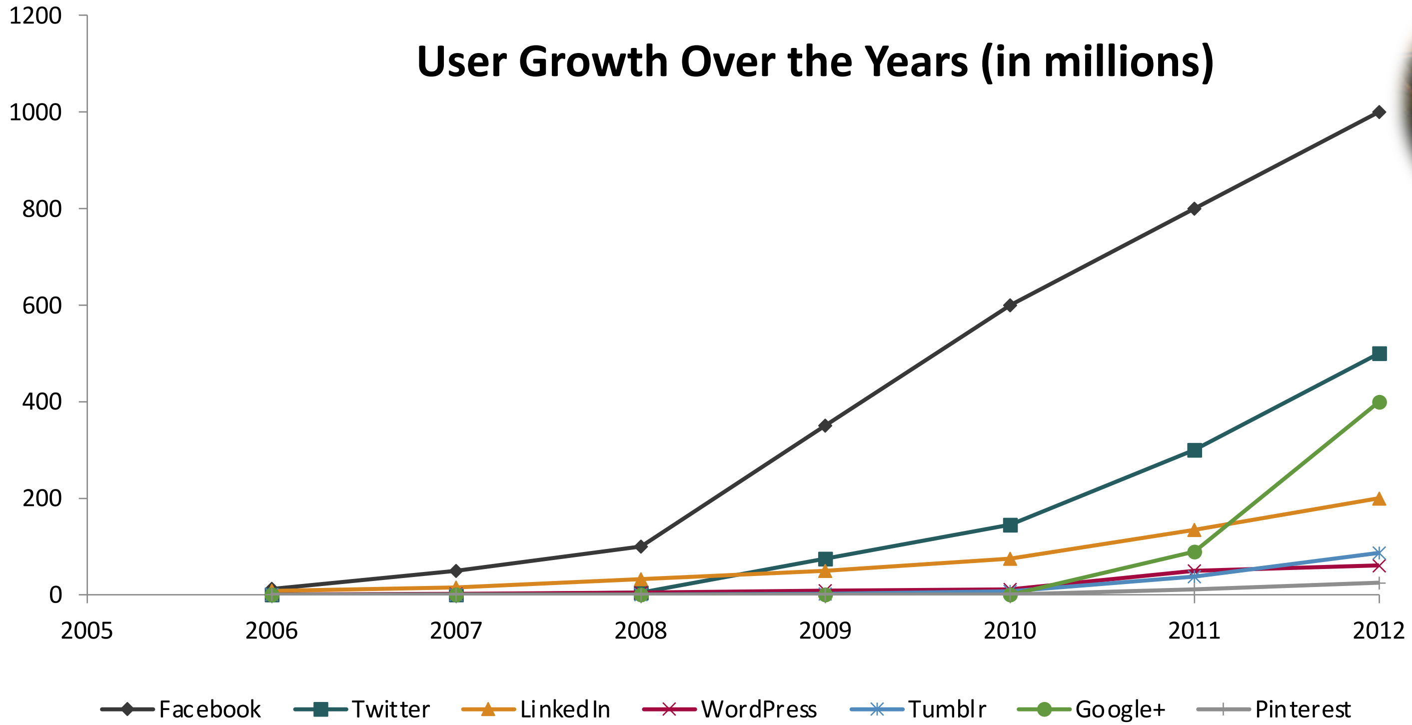
SAP ERP

WhatsApp

450 million active users

> 50 billion messages every day

Rapid Growth and
Varying Usage Patterns

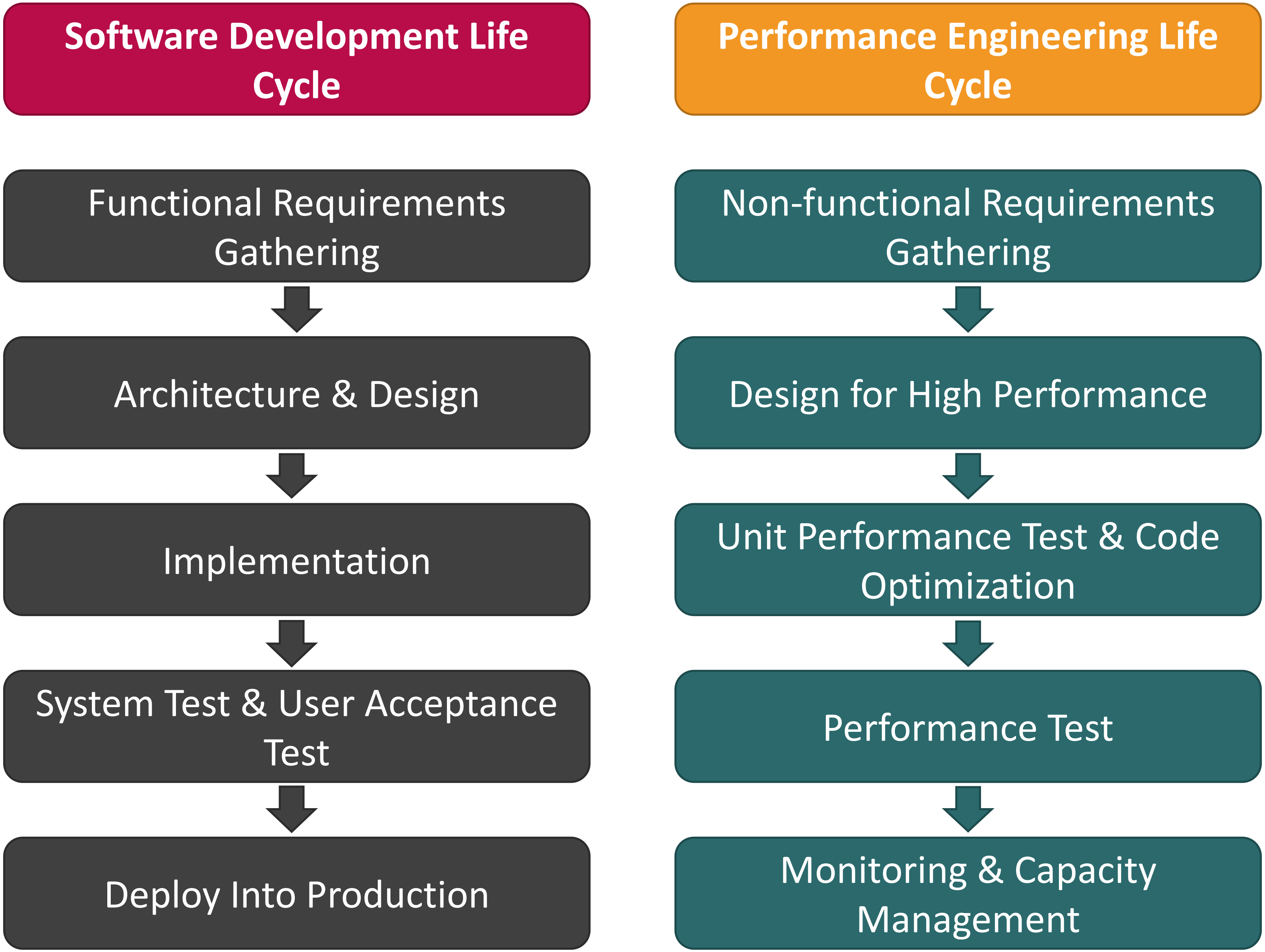


Failures of large-scale systems are often due to performance issues rather than functional bugs

A page load slowdown of only one second could cost $1.6 billion

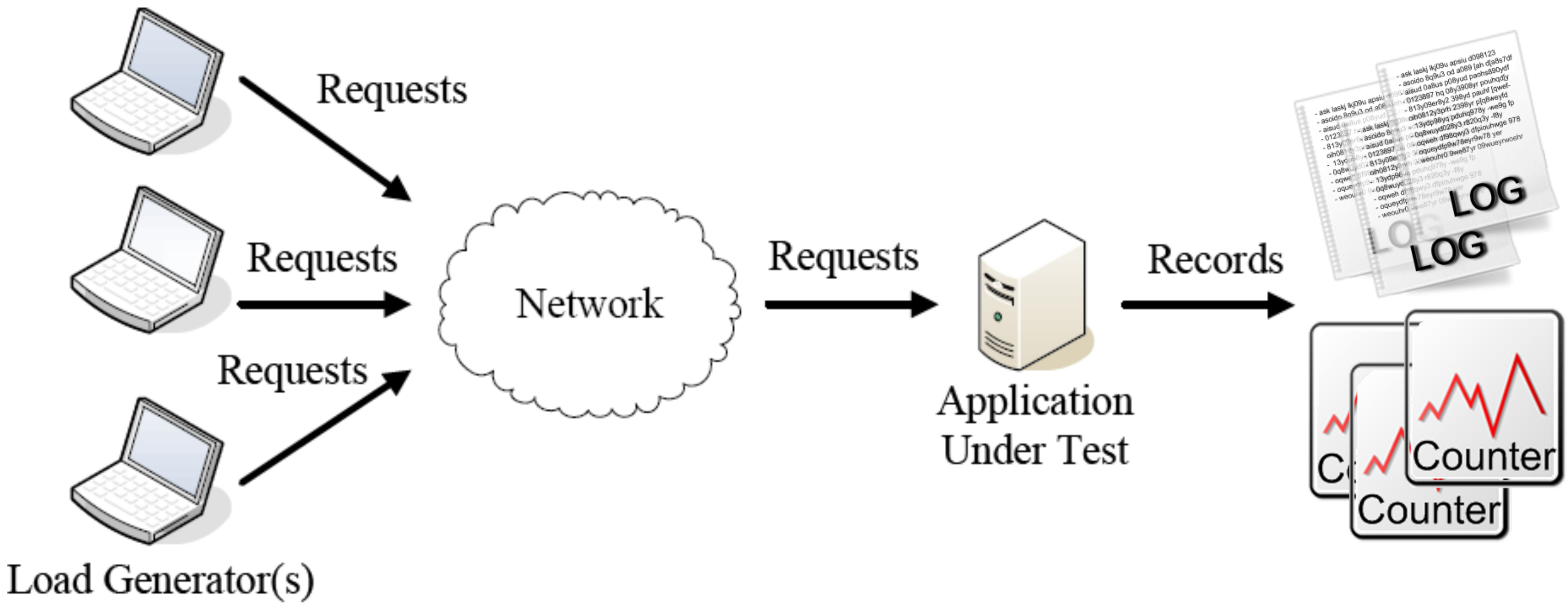
Software Performance Engineering (SPE)

- The set of tasks or activities that need to be performed across the Software Development Life Cycle (SDLC) to meet the documented Non-Functional Requirements (Performance, Scalability, Availability, Reliability, etc.)



Source: https://tangowhisky37.github.io/PracticalPerformanceAnalyst/pages/spe_fundamentals/performance_engineering_101/

Performance Testing



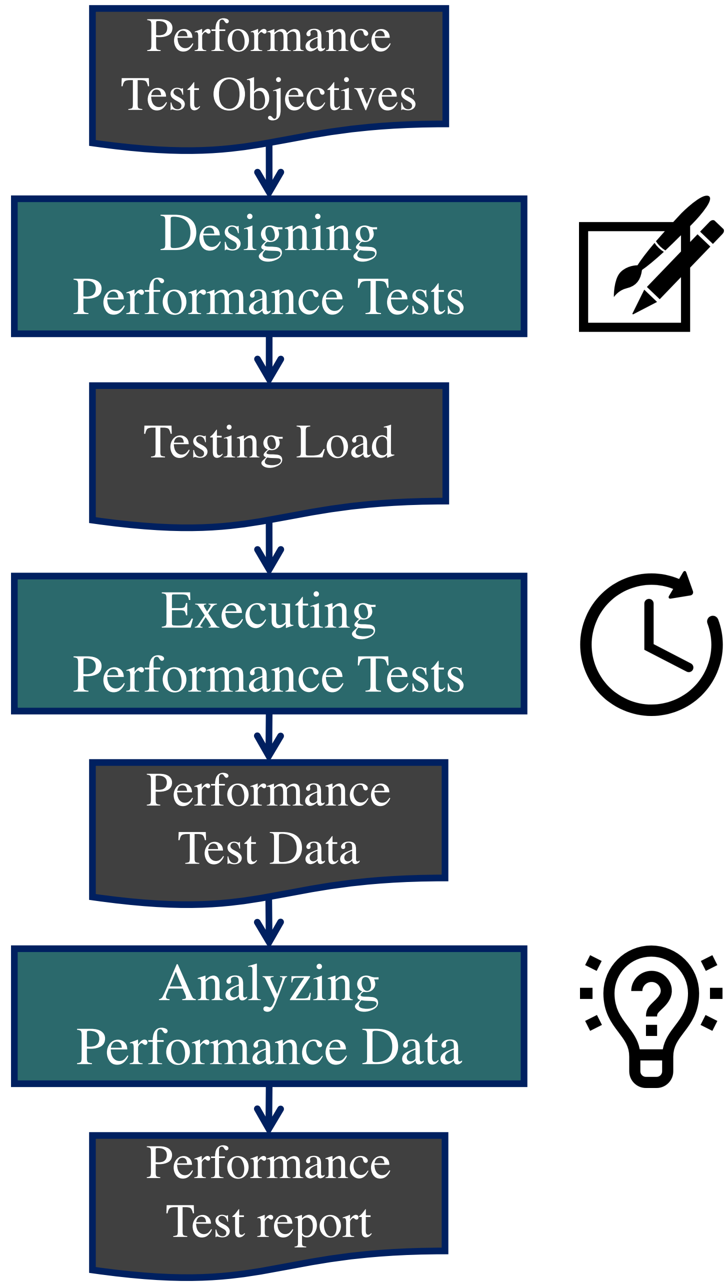
Test Design **Test Execution** **Test Analysis**

Mimics multiple users repeatedly performing the same tasks

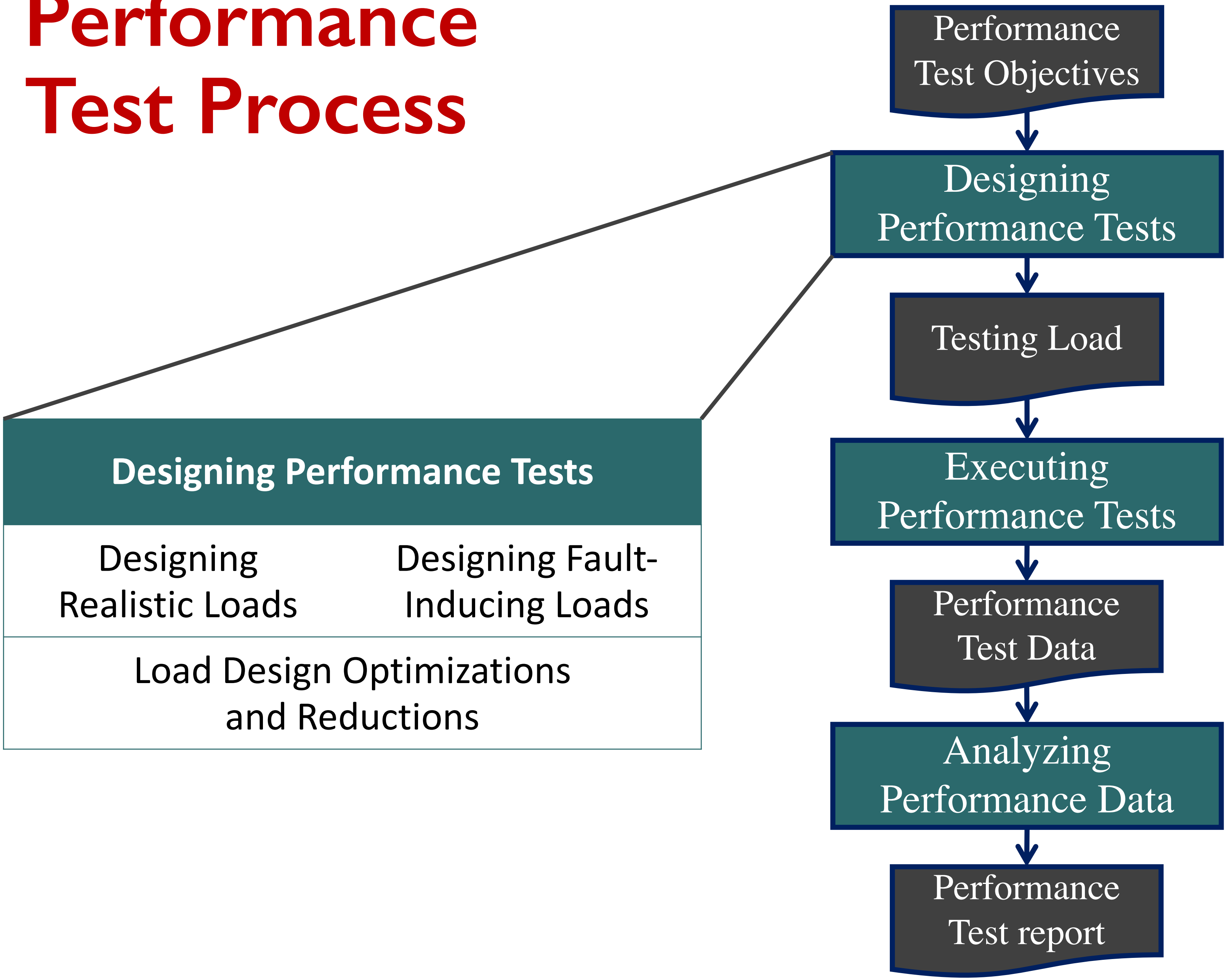
Take hours or even days

Produces GB/TB of data that must be analyzed

Performance Test Process



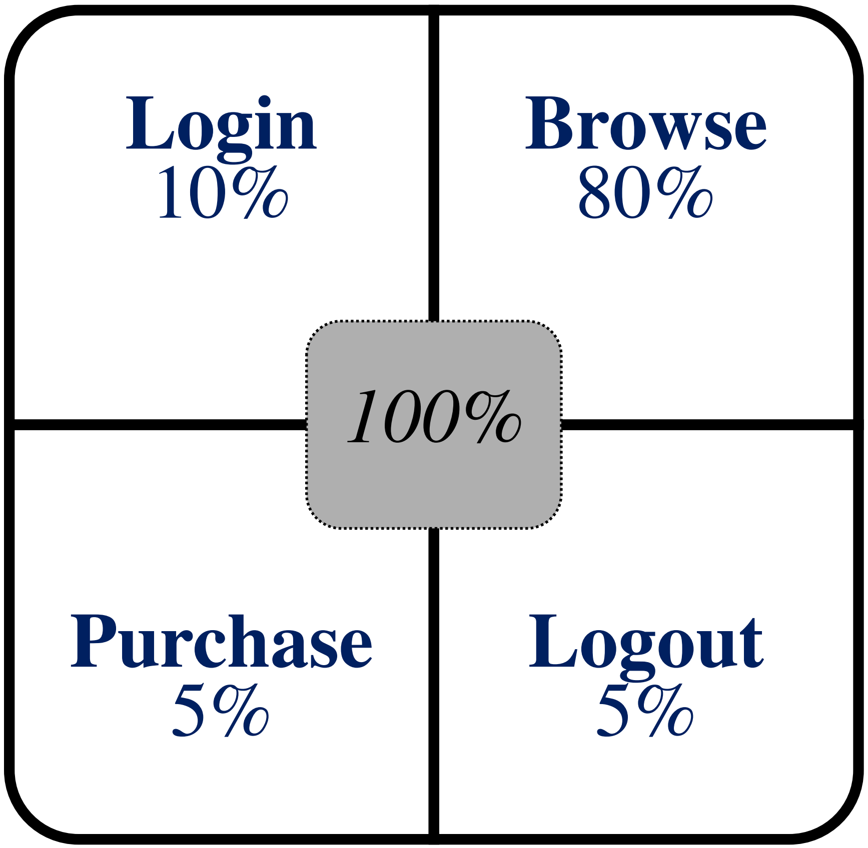
Performance Test Process



Designing Realistic Loads

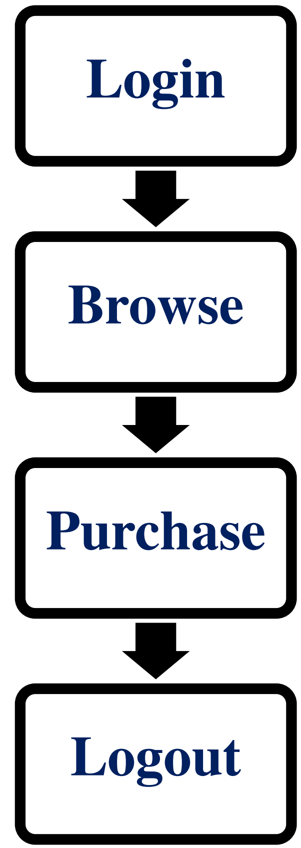
An E-Commerce System

Aggregate Workload



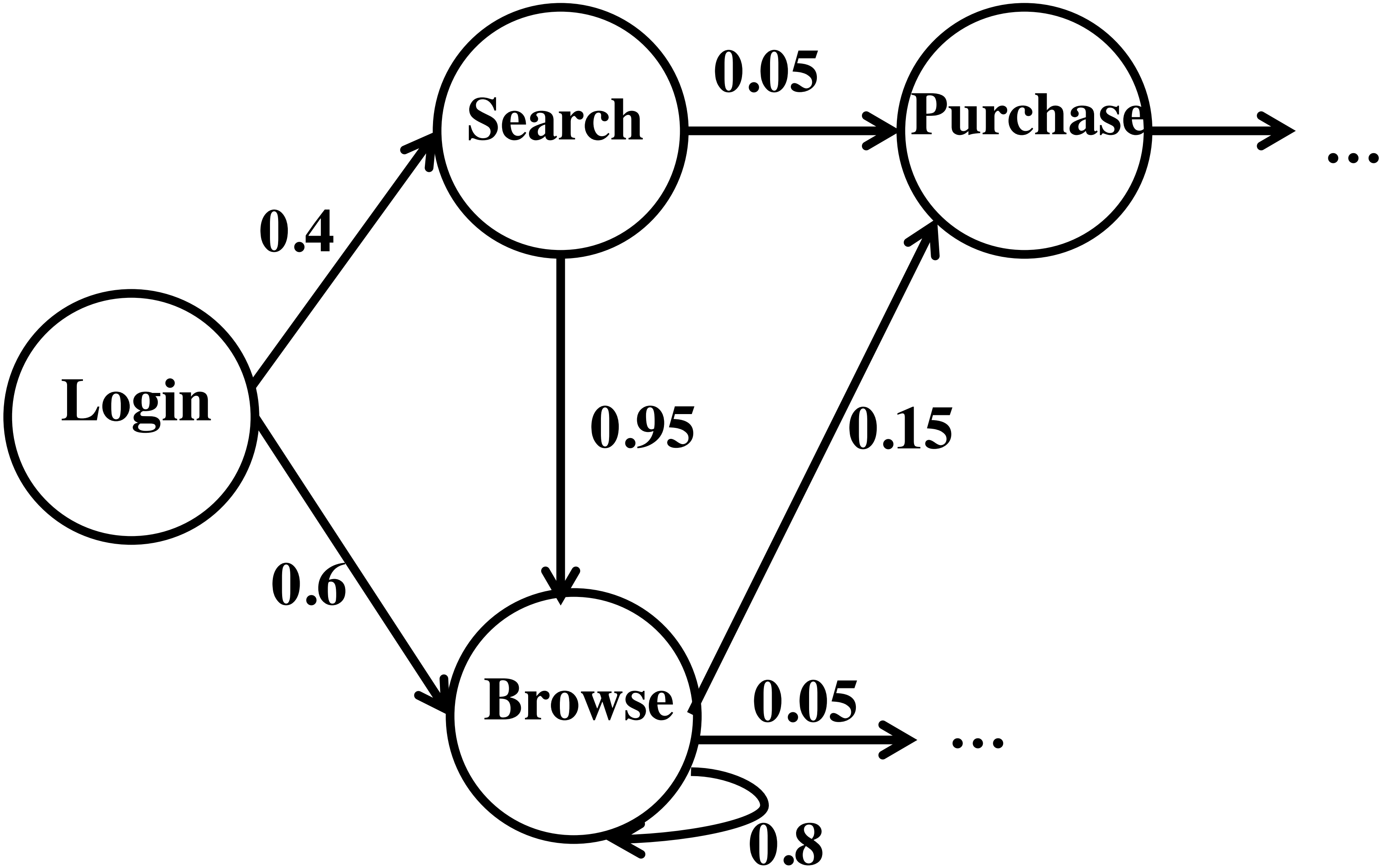
Steady Load, Step-wise load, Extrapolated load

Use-Case



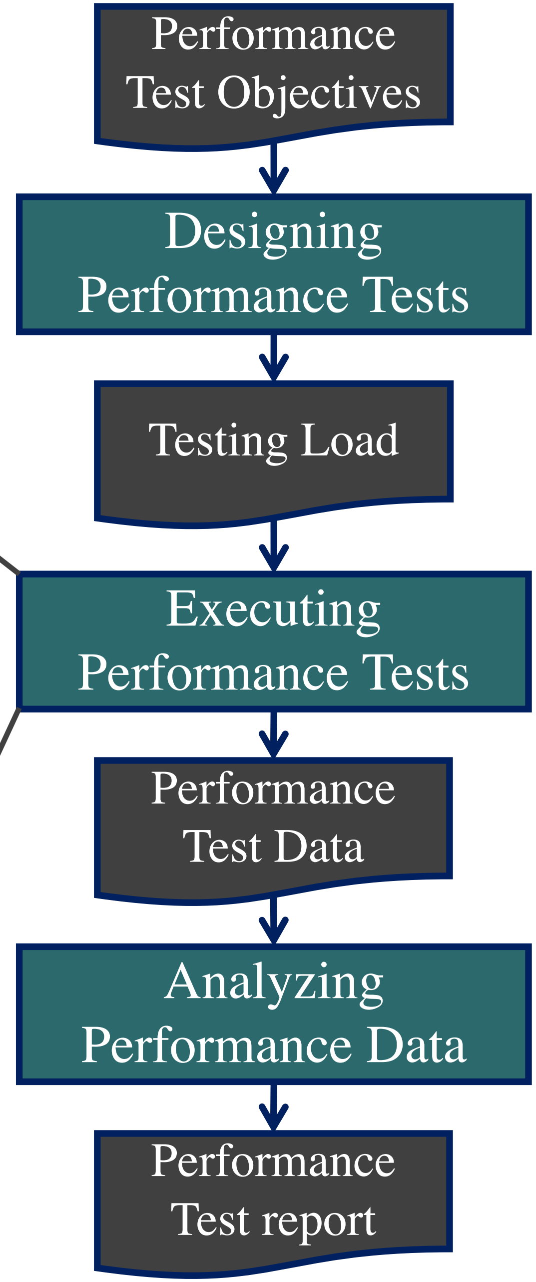
Load Derived from UML, Markov and Stochastic Form-oriented Models

Use-Case (2)
- Markov Chain



Performance Test Process

Executing Performance Tests		
Live-user Based Execution	Driver Based Execution	Emulation Based Execution
Setup		
Load Generation and Termination		
Test Monitoring and Data Collection		



Live-user Based Test Execution

- Reflects realistic user behavior
- Obtain real user feedbacks on acceptable performance and functional correctness

- Hard to scale (e.g., limited testing time)
- Limited test complexity due to manual coordination

- Coordinated live-user testing
- Users are selected based on different testing criteria (e.g., locations, browser types, etc.)

Driver-based Test Execution

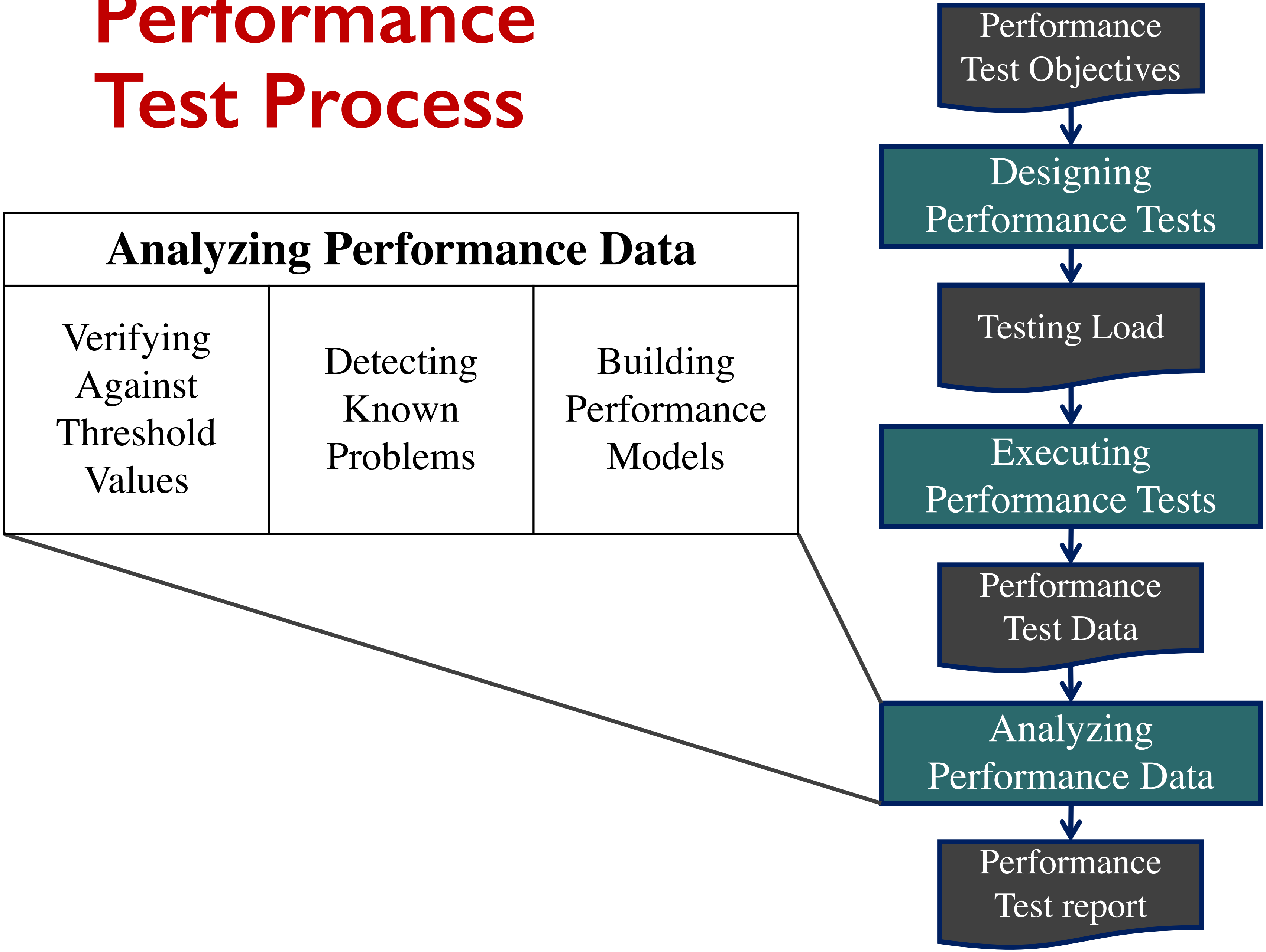
- 👍 Easy to automate
- 👍 Scale to large number of requests

- 👎 Load driver configurations
- 👎 Hard to track some system behavior (e.g., audio quality or image display)

- Specialized Benchmarking tools (e.g., LoadGen)
- Centralized Load Drivers (e.g, LoadRunner, WebLoad)
  - Easy to control load, but hard to scale (limited to a machine's memory)
- Peer-to-peer Load Drivers (e.g., JMeter, PeerUnit)
  - Easy to scale, but hard to control load

APACHE JMeter™

Performance Test Process



Sample Counters

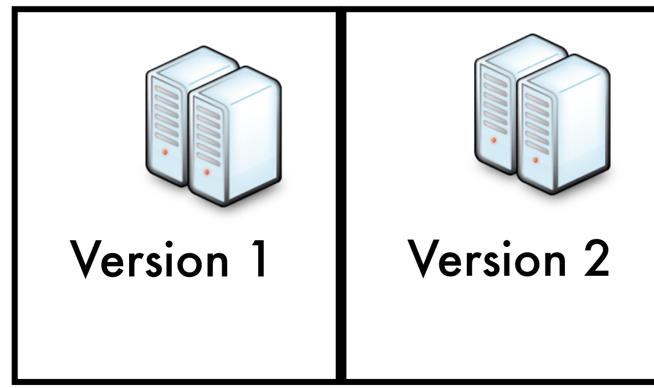
	A	B	C	D	E
1	Time	Disk Reads/sec	Disk Writes/sec	Page Faults/sec	Memory
2	2/29/08 16:58	0.049986394	0.000723659	0.003876542	3534848
3	2/29/08 17:01	0	0	0	3534848
4	2/29/08 17:04	0.060612225	0.027551011	0.016530607	3534848
5	2/29/08 17:07	0	0	0	3534848
6	2/29/08 17:10	0	0	0	3534848
7	2/29/08 17:13	0.060733302	0.027606046	0.016563628	3534848
8	2/29/08 17:16	0	0	0	3534848
9	2/29/08 17:19	0.060727442	0.027603383	0.01656203	3534848
10	2/29/08 17:22	0	0	0	3534848
11	2/29/08 17:25	0	0	0	3534848
12	2/29/08 17:28	0	0	0	3534848
13	2/29/08 17:31	0	0	0	3534848
14	2/29/08 17:34	0.121368621	0.055167555	0.038617289	3534848
15	2/29/08 17:37	0	0	0	3534848
16	2/29/08 17:40	0	0	0	3534848
17	2/29/08 17:43	0	0	0	3534848
18	2/29/08 17:46	0	0	0	3534848
19	2/29/08 17:49	0	0	0	3534848
20	2/29/08 17:52	0	0	0	3534848
21	2/29/08 17:55	0.121392912	0.055178596	0.033107158	3534848
22	2/29/08 17:58	0.060592703	0.027542138	0.02203371	3534848

Sample Execution Logs

#	Log Lines
1	time=1, thread=1, session=1, receiving new user registration request
2	time=1, thread=1, session=1, inserting user information to the database
3	time=1, thread=2, session=2, user=Jack, browse catalog=novels
4	time=1, thread=2, session=2, user=Jack, sending search queries to the database
5	time=3, thread=1, session=1, user=Tom, registration completed, sending confirmation email to the user
6	time=3, thread=2, session=2, database connection error: session timeout
7	time=4, thread=1, session=1, fail to send the confirmation email, number of retry = 1
8	time=6, thread=2, session=2, user=Jack, successfully retrieved data from the database
9	time=7, thread=2, system health check
10	time=8, thread=1, session=1, registration email sent successfully to user=Tom
11	time=9, thread=2, session=3, user=Tom, browse catalog=travel
12	time=10, thread=2, session=3, user=Tom, sending search queries to the database
13	time=10, thread=3, session=4, user=Jim, updating user profile
14	time=11, thread=3, session=4, user=Jim, database error: deadlock

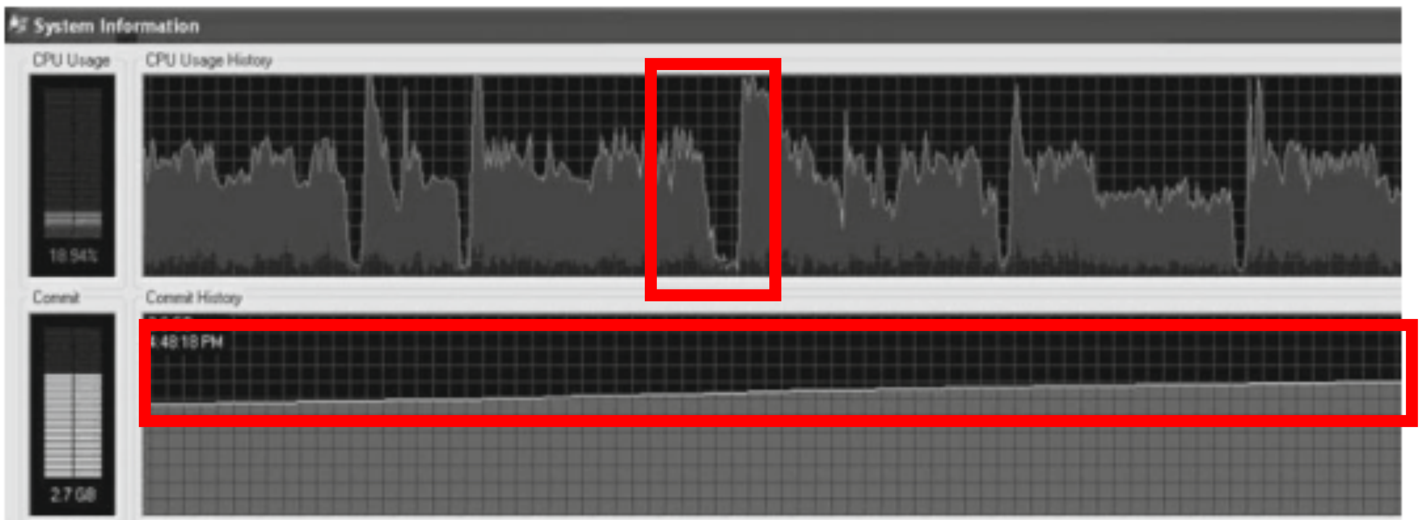
Verifying Against Threshold Values

Threshold from requirement



Threshold from a prior version

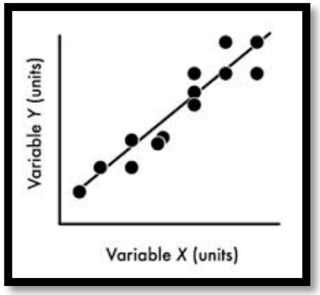
Looking for known patterns: Deadlocks and memory leak



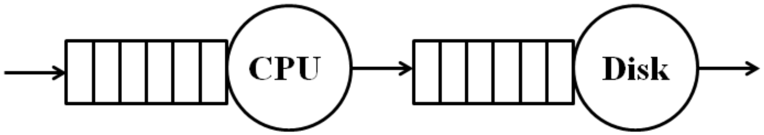
Performance data under **steady** load

[Avritzer et al., 2012]

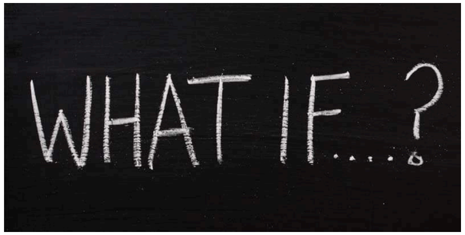
Building performance models

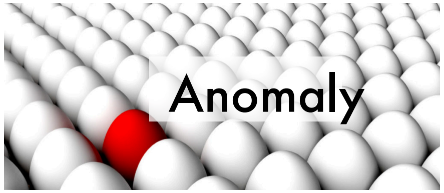


Machine learning (Black box) model



Queuing (white box) Model





Profiling

- A form of dynamic program analysis that measures the complexity of the program in terms of space (memory) or time, or the frequency and duration of function calls.
- Its objective is the **optimization of the program** and the management of resources.
- It is a process that helps to understand the behavior of a program.
- It also helps evaluate and compare performance of different architectures.
- Profiling has two important components: instrumentation and sampling.

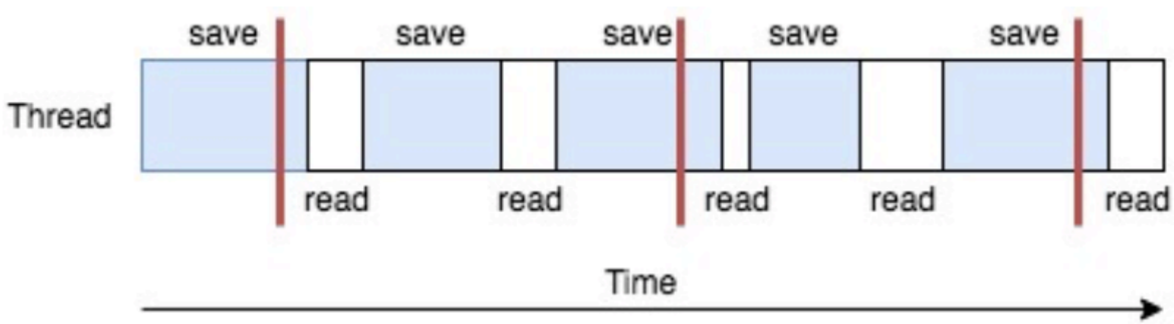
Profiling: Instrumentation

- It is possible to collect data by external tools, but this data is not detailed enough and of a sufficient level of granularity.
- For this reason, instrumentation is used.
  - A technique that adds code (probes) in the monitored program to collect performance data.
- It is possible to add probes at several levels of the system.
  - Source code (manually or automatically)
  - Assisted by the compiler
  - Binary code

- Motivation for profiling:
  - Collect exactly the data needed and infer the locality of the data.
  - Control the granularity of data.
  - Control the measurement process by activating and deactivating probes.

Profiling: Sampling

- Sampling does not affect the execution of the program.
  - No instruction is inserted in the source elbow nor in the compiled code.
- The operating system suspends the CPU at regular intervals and the profiler records the instruction that is currently executing.
- The profiler correlates the instruction with the corresponding point in the code.
- The profiler returns the frequency of execution of code points.
- Repeat profiling with sampling several times to obtain statistical significance.



Profiling: Automated Profiling

- Automated profiling facilitates optimization and guarantees continuous integration and continuous quality assurance.
- It also reduces optimization costs.
- Profiling tools are able to calculate a large number of measurements and produce detailed reports.
- Warning! Some profiling methods are characterized as intrusive, which can affect the results of the process.

Profiling vs Performance testing

Profiling → Performance testing

- We can use profiling to understand the behavior of our program ...
- ... and identify the use of resources (CPU, memory etc.)
- After that, we can define the thresholds and objectives for the performance and test them.
- We can also train or provide inputs for our performance models.

Performance testing → Profiling

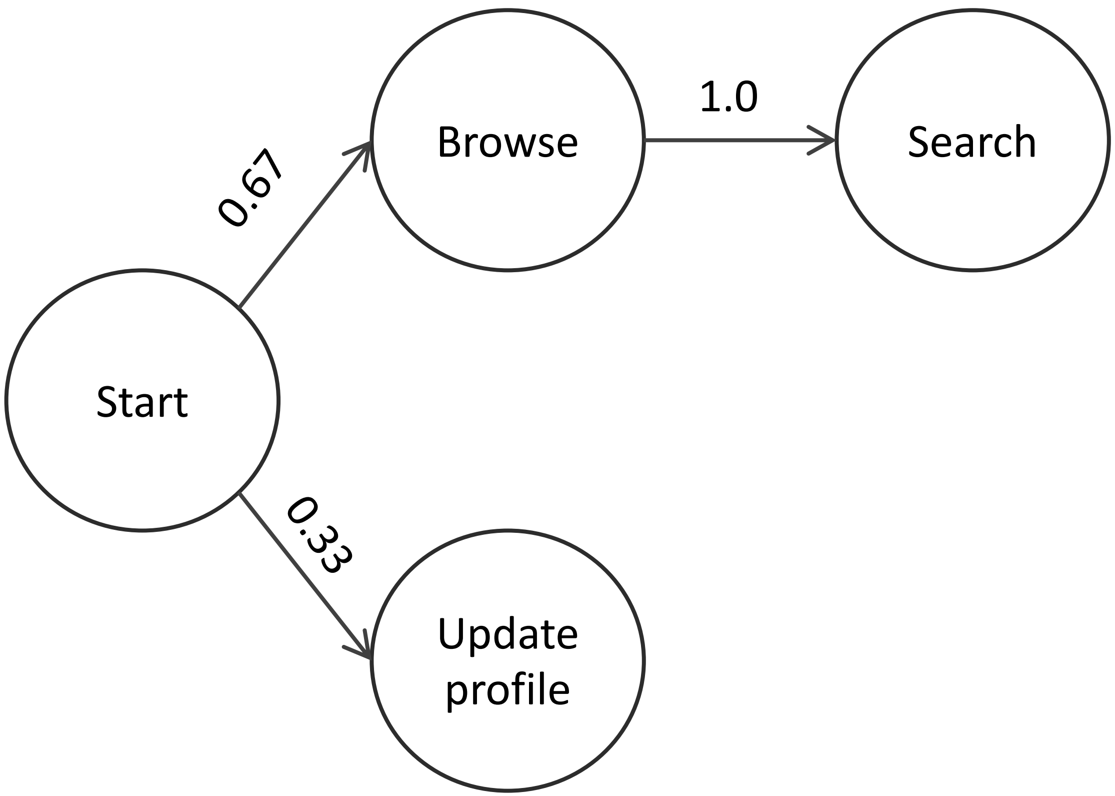
- Testing will indicate the presence of performance issues.
- According to this indication, profiling will reveal the exact point of the bottleneck.
- After, we optimize the code and rerun the tests.

Exercise from last class: Design realistic loads for performance testing

- Based on the following sample logs, design a use-case model using the Markov chain

#	Log Lines
1	time=1, thread=1, session=1, receiving new user registration request
2	time=1, thread=1, session=1, inserting user information to the database
3	time=1, thread=2, session=2, user=Jack, browse catalog=novels
4	time=1, thread=2, session=2, user=Jack, sending search queries to the database
5	time=3, thread=1, session=1, user=Tom, registration completed, sending confirmation email to the user
6	time=3, thread=2, session=2, database connection error: session timeout
7	time=4, thread=1, session=1, fail to send the confirmation email, number of retry = 1
8	time=6, thread=2, session=2, user=Jack, successfully retrieved data from the database
9	time=7, thread=2, system health check
10	time=8, thread=1, session=1, registration email sent successfully to user=Tom
11	time=9, thread=2, session=3, user=Tom, browse catalog=travel
12	time=10, thread=2, session=3, user=Tom, sending search queries to the database
13	time=10, thread=3, session=4, user=Jim, updating user profile
14	time=11, thread=3, session=4, user=Jim, database error: deadlock

Exercise from last class: Design realistic loads for performance testing



Today: Software Performance Models

- Software Performance Modeling (SPM)
- Execution graphs
- Queuing Networks
- Machine learning based performance models

- References:

  - Jain, Raj. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing, 1991.
  - Gao, Ruoyu, et al. *A framework to evaluate the effectiveness of different load testing analysis techniques.* ICST '16.
  - Connie U. Smith. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addision-Wesley. 2001.
  - Kaushal Kumar's lecture slides for course *Software Performance Analysis* at Queen's University: https://research.cs.queensu.ca/home/elgazzar/soft437/

Software Performance Models (SPM)

- Formal representations of the software to capture aspects and information of the performance.
- Built as early as design and architecture models to express and understand the non-functional requirements.

Software Performance Models (SPM)

- They allow us to:
  - ✓Estimate the performance of the software.
  - ✓Estimate resource needs.
  - ✓Identify performance issues as early as possible.
  - ✓Simulate the execution of the software under certain conditions (number of users and size of the infrastructure).
  - ✓Establish software performance for medium, best, and worst case scenarios.

Software Performance Models (SPM)

- SPM sometimes provide a graphical representation of the system's execution that matches its structure.
  - It is possible to produce the performance models by transforming the design models.
- While the design models of the system capture the static aspects, the performance models capture the dynamic aspects.
- Types of performance models:
  - Software Execution Models
  - Queuing Networks (System Execution Models)
  - Machine learning based models
  - Others (e.g., Stochastic Petri Nets)

Software Execution Models

Software Execution Models

- Constructed early in the development process to ensure that the chosen software architecture can achieve the required performance objectives
- Captures essential performance characteristics of the software
- Provides a static analysis of the mean, best, and worst-case response time
- Characterizes the resource requirements of the proposed software alone, in the absence of other workloads, multiple users or delays due to contention for resources

Software Execution Model (con't)

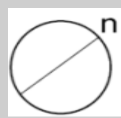
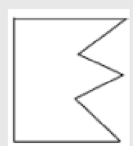
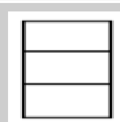
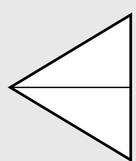
- Software execution models are generally sufficient for identifying serious performance problems at the architectural and early design phases
- We can refine software execution model in the critical areas

The absence of problems in the software model does not mean that there are none

Execution graphs

- Execution graphs are one type of software execution model.
- The graphs represent the execution (a sequence of operations) of the system.
  - ✓ An execution graph is constructed for each performance scenario.
- Execution graphs are not sufficient for a complete analysis of software performance, but they work well for understanding the software and its non-functional requirements.
  - ✓ The special annotation can give us an idea of the performance.
  - ✓ We can combine the graphs with other models (like QN we will see) to complement the analysis.

Graph notation

Node types	Graph notation	Description
Basic nodes		Represent processing steps at the lowest level of detail that is appropriate for the current development stage
Expanded nodes		Represent processing steps elaborated in another subgraph
Repetition nodes		Subsequent nodes are repeated n times: the last node has an edge to the repeat node
Case node		Represent conditional execution of processing steps; each attached node has a probability of execution
Pardo node		Attached nodes run in parallel: All nodes must complete (join) before proceeding.
Division node		Attached nodes represent new processing threads; they need not all complete before proceeding.

Example: General ATM Scenario

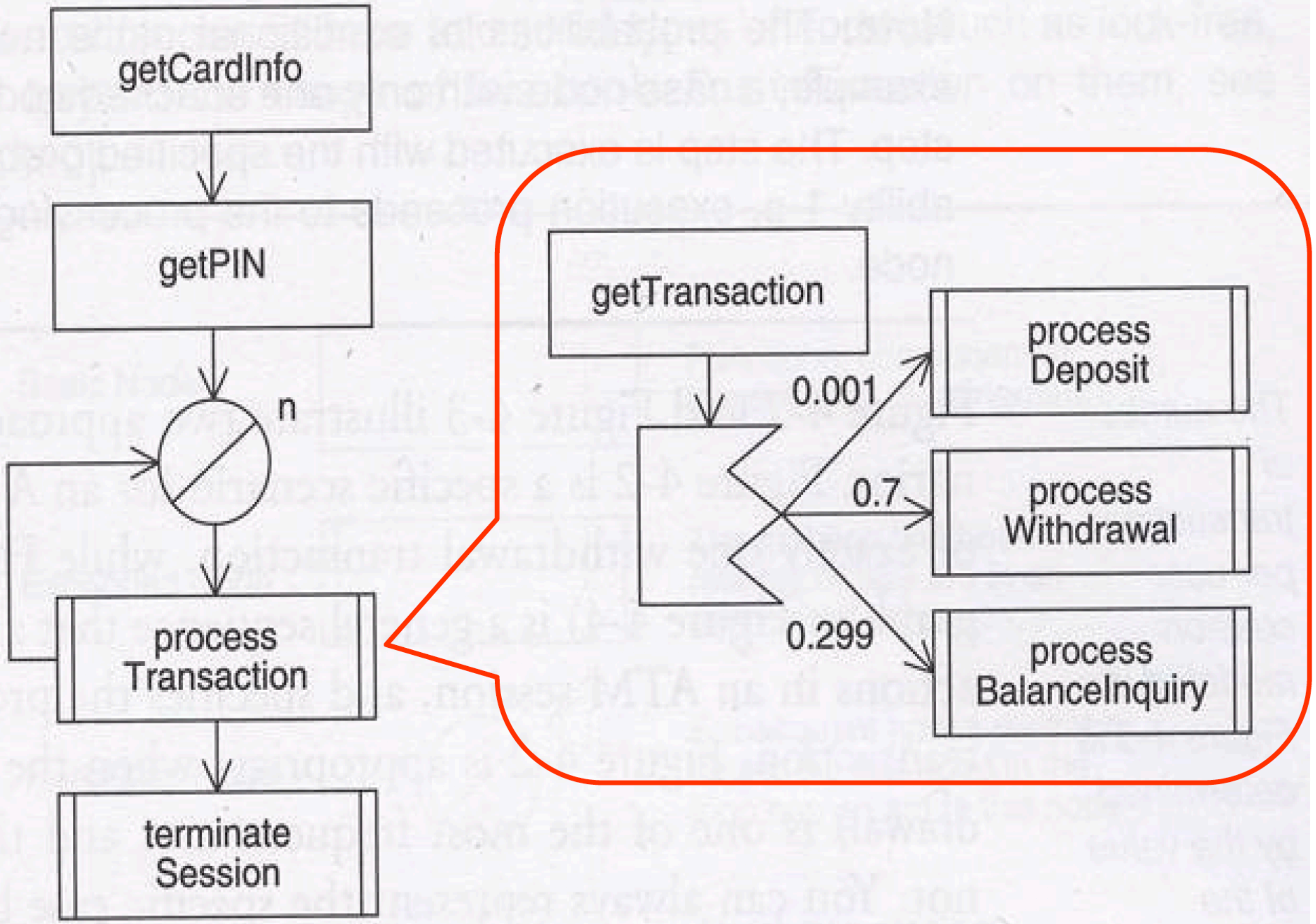


Figure 4-3: Execution Graph for General ATM Scenario

Figure 4-4: Subgraph for processTransaction Expanded Node

From book: *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*

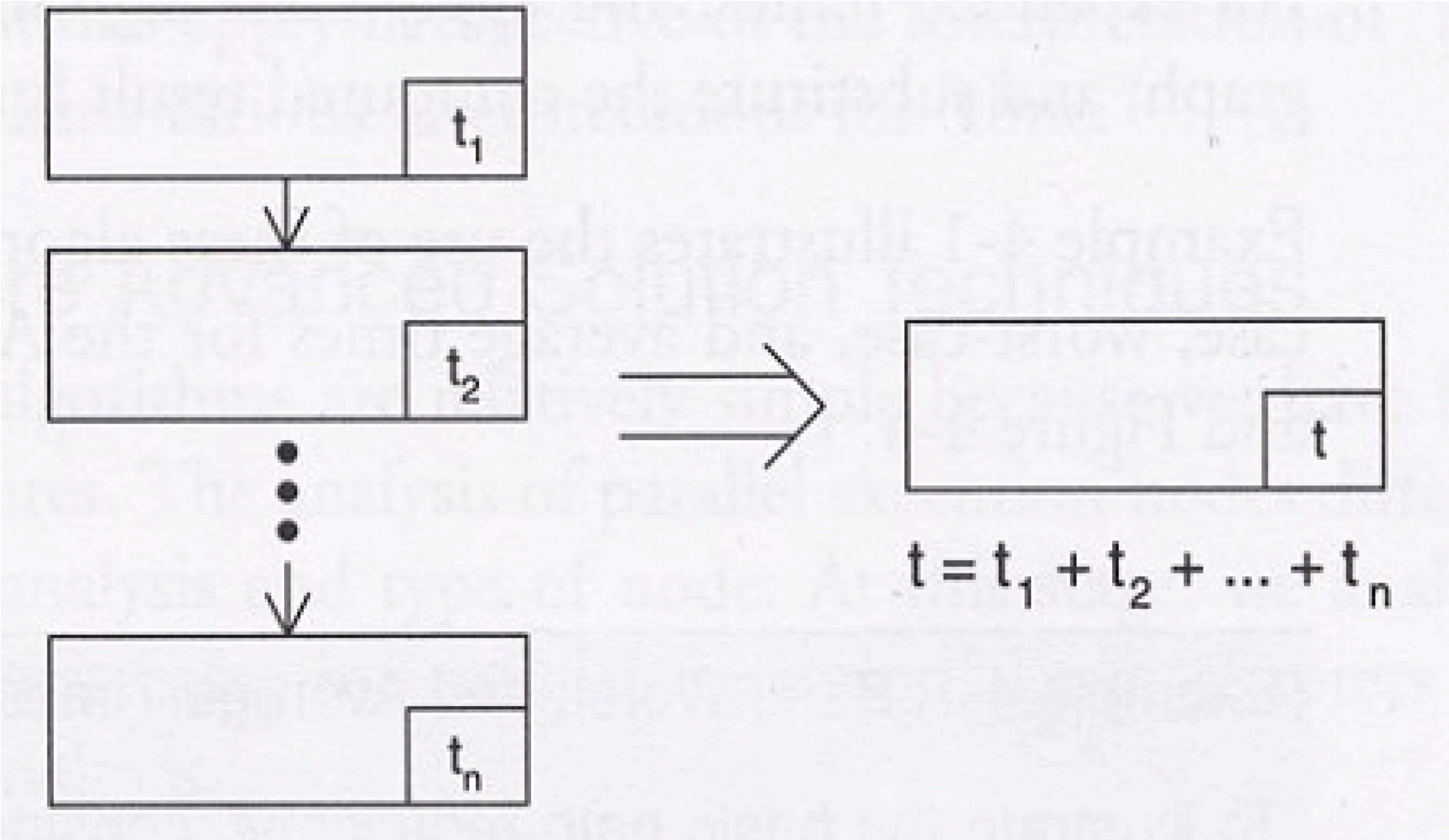
Software Execution Model Analysis

- Primary purposes of software execution model analysis are
  - Make a quick check of the best-case response time in order to ensure the architecture and design will lead to satisfactory performance
  - Assess the performance impact of alternatives
  - Identify critical parts of the system for performance management
  - Derive parameters for the system execution model
- The algorithms are formulated for evaluating graphs

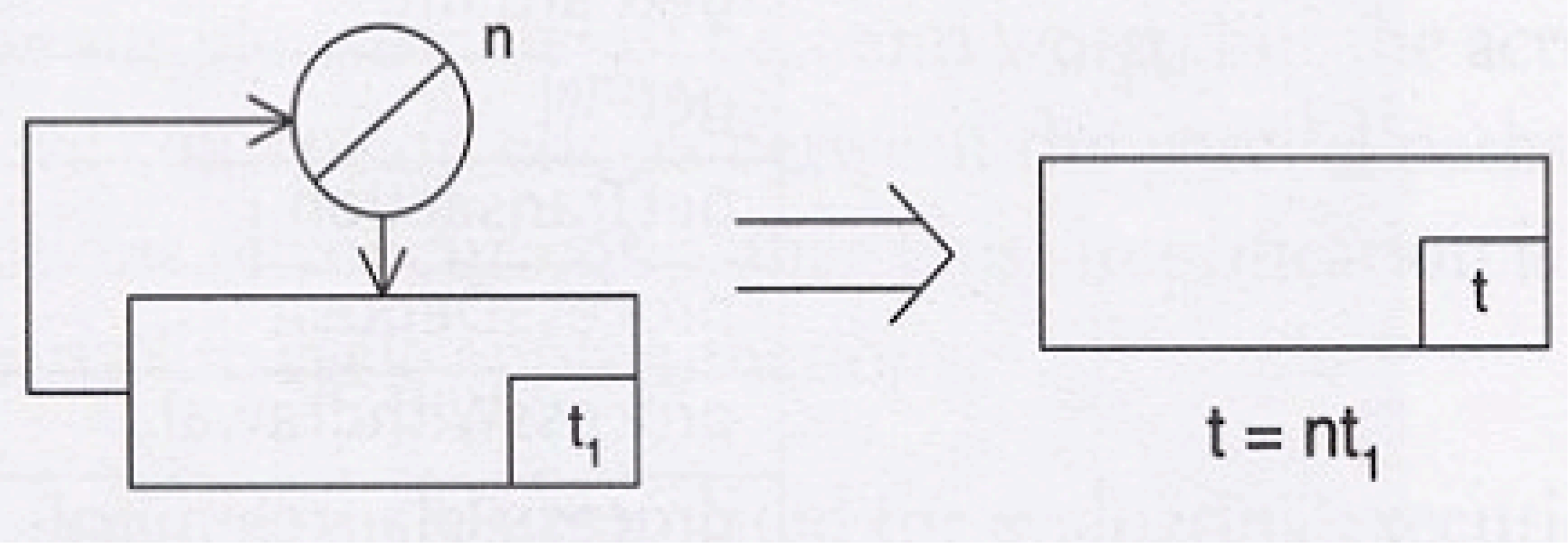
Basic Solution Algorithms

- The algorithms are ‘easy’ to understand
  - Examine graphs and identify a basic structure
  - Compute the time of a basic structure and reduce the basic structure to a ‘computed node’
  - Continue until only one node left
- Basic structures are
  - Sequences
  - Loops
  - Cases

Graph Reduction for Sequential Structures



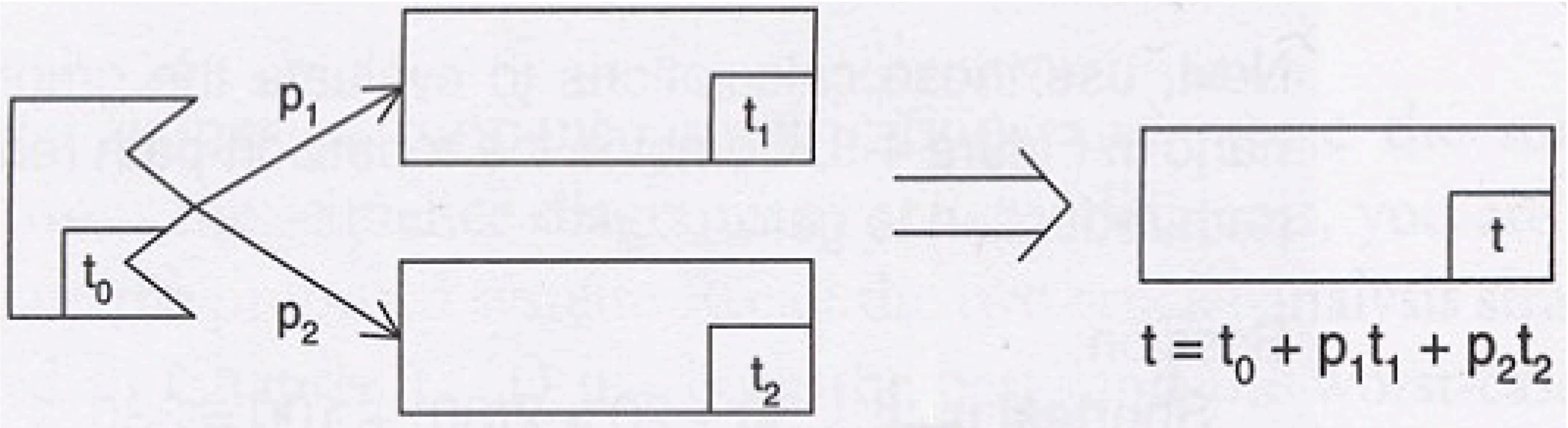
Graph Reduction for Loop Structures



Graph Reduction for Case Nodes

- The computation for case nodes differs for shortest path, longest path, and average analyses
  - Shortest path: the time for the case node is the minimum of the times for the conditionally executed nodes
  - Longest path: the time for the case node is the maximum of the times for the conditionally executed nodes
  - For the average analysis: the time is multiplying each node's time by its execution probability

Graph Reduction for Case Nodes



Exercise: ATM Scenario

What's the best, worst, and average execution time?

To illustrate the basic path reductions, consider the ATM scenario in Figure 4-3 and the subgraph for processTransaction in Figure 4-4. Assume the node "times" in the following table.

Node	Time
getCardInfo	50
getPIN	20
getTransaction	30
processDeposit	500
processWithdrawal	200
processBalanceInquiry	50
terminateSession	100

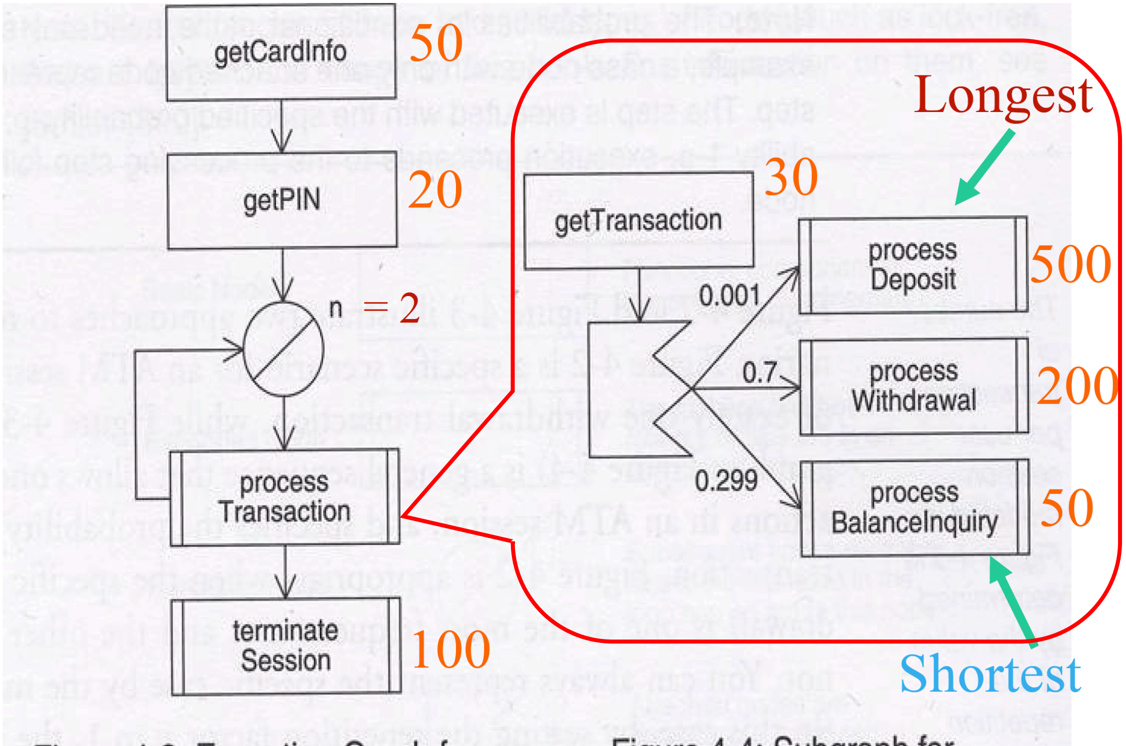


Figure 4-3: Execution Graph for General ATM Scenario

Figure 4-4: Subgraph for processTransaction Expanded Node

Analysis Procedures

- Use both the best-and the worst-case estimates of resource requirements for each basic node
- Begin with a simplistic analysis of the best case and introduce more sophisticated analyses of realistic cases as more detailed information becomes available

Software Resource Requirements

- Each basic node has specified SW resource requirements $\mathbf{A}_j$ for each service unit j, e.g.

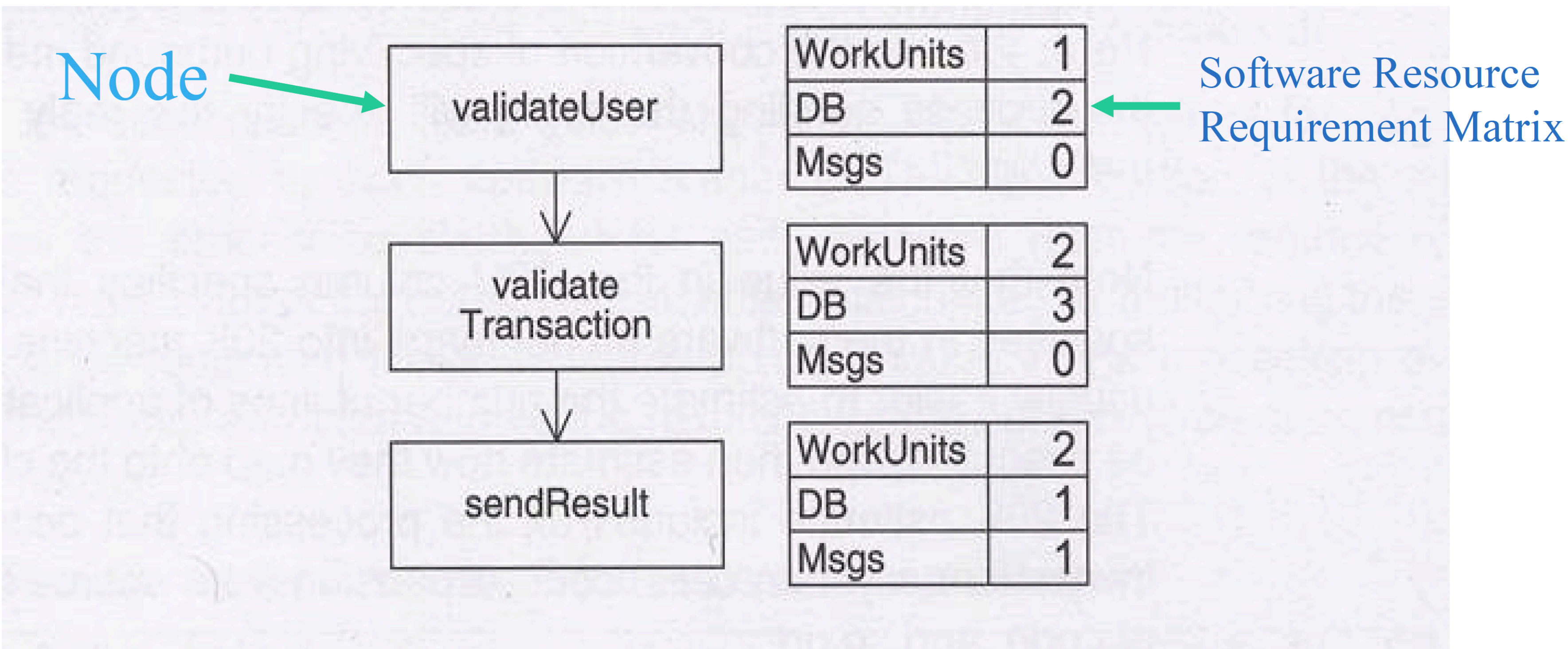


Figure 4-12: authorizeTransaction Software Resource Requirements

Processing Overhead Matrix

- A chart of the computer resource requirements for each of the software resource requests

Table 4-1: Processing Overhead

Device	CPU	Disk	Network
Quantity	1	1	1
Service Unit	KInstr.	Phys. I/O	Msgs.
WorkUnit	20	0	0
DB	500	2	0
Msgs	10	2	1
Service time	0.00001	0.02	0.01

Hardware Resources

Mapping between software resource requirements and computer device usage

Computing the total execution time

- ***STEP 1:*** uses the processing overhead matrix to calculate the total computer resources required per software resource for each node in the graph

Total Computer Resource Requirements for sendResult

Software Resource Requests		Processing Overhead		
Name	**Service Units**	**CPU Kinstr**	**Physical I/O**	**Network Messages**
WorkUnit	2	20	0	0
DB	1	500	2	0
Msgs	1	10	2	1
Total: sendResult		550	4	1

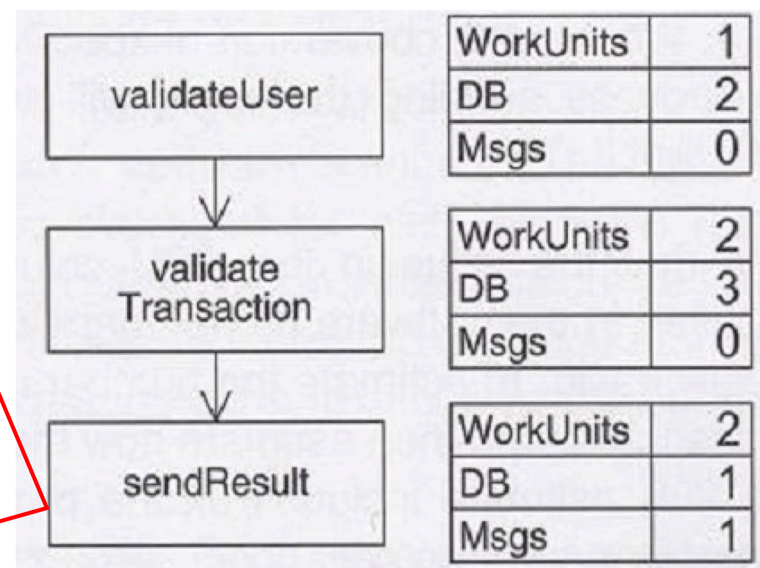
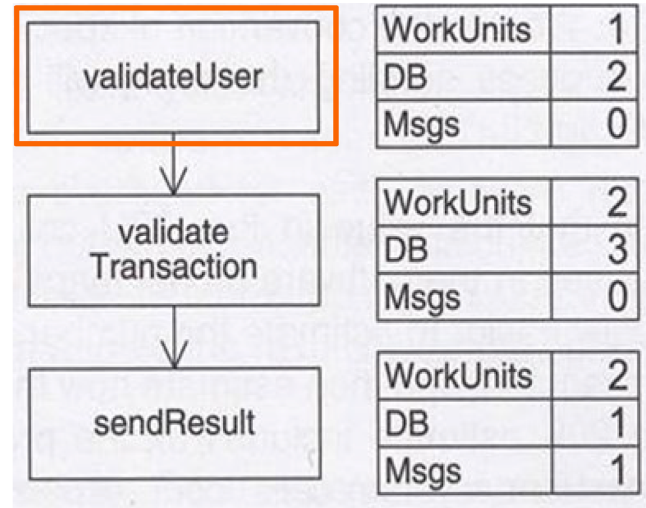


Table 4-1: Processing Overhead

Device	CPU	Disk	Network
Quantity	1	1	1
Service Unit	KInstr.	Phys. I/O	Msgs.
WorkUnit	20	0	0
DB	500	2	0
Msgs	10	2	1
Service time	0.00001	0.02	0.01



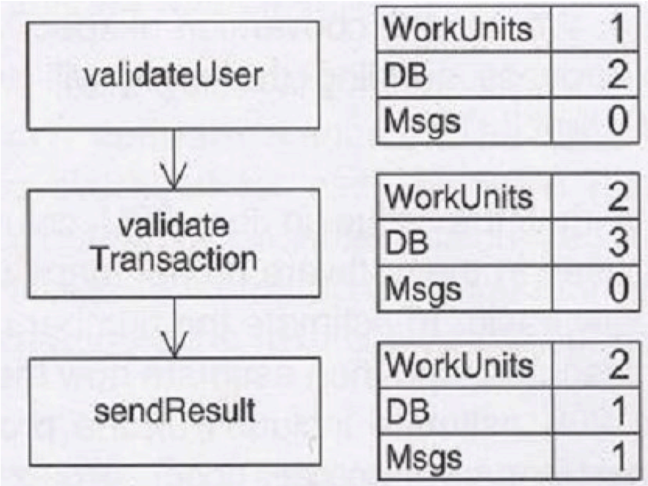
Software Resource Requirements

		KInstr.	Phys. I/O		Msgs.
WorkUnits	1	20	0		0
DB	2	500	2		0
Msgs	0	10	2		1
validateUser		**1020**	**4**		**0**
		0.00001	**0.02**		**0.01**

Computing the total execution time

- ***STEP 2:*** computes the total computer resource requirements for the graph

Processing Step	CPU Kinstr	Physical I/O	Network Messages
validateUser	1,020	4	0
validateTransaction	1,540	6	0
sendResult	550	4	1
Total: authorizeTransaction	3,110	14	1



Computing the total execution time

- ***STEP 3:*** compute the best-case elapsed time

Table 4-1: Processing Overhead

Device	CPU	Disk	Network
Quantity	1	1	1
Service Unit	KInstr.	Phys. I/O	Msgs.
WorkUnit	20	0	0
DB	500	2	0
Msgs	10	2	1
Service time	0.00001	0.02	0.01

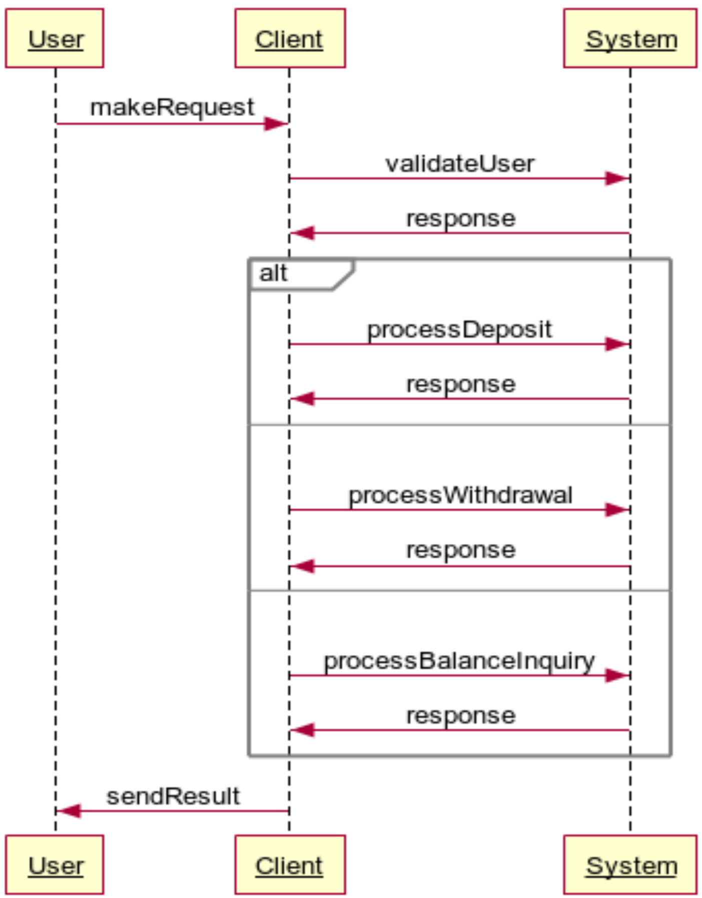
Processing Step	CPU Kinstr	Physical I/O	Network Messages
validateUser	1,020	4	0
validateTransaction	1,540	6	0
sendResult	550	4	1
Total: authorizeTransaction	3,110	14	1

3,110*0.00001 + 14*0.02 + 1*0.01 = **0.32**

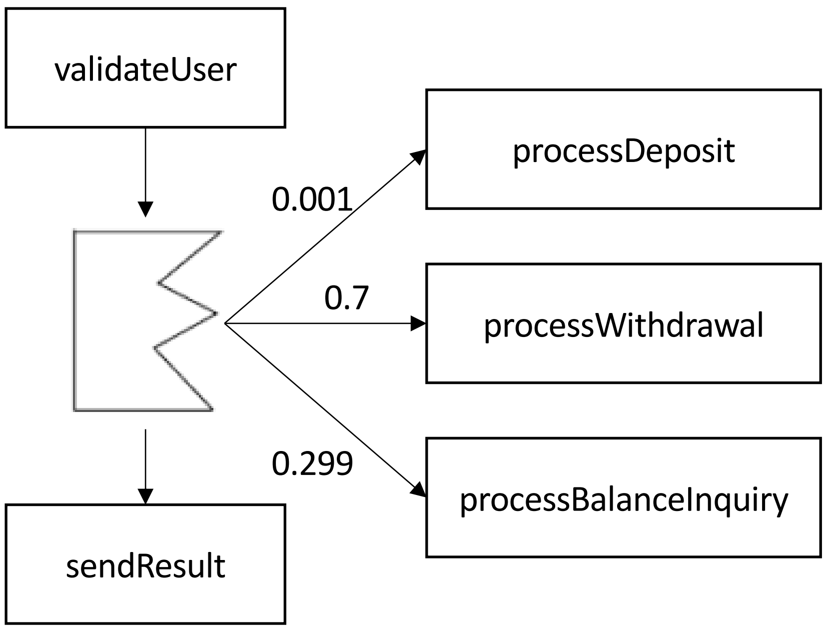
Exercise: processTransaction scenario

Estimate the best, worst, and average execution time

UML

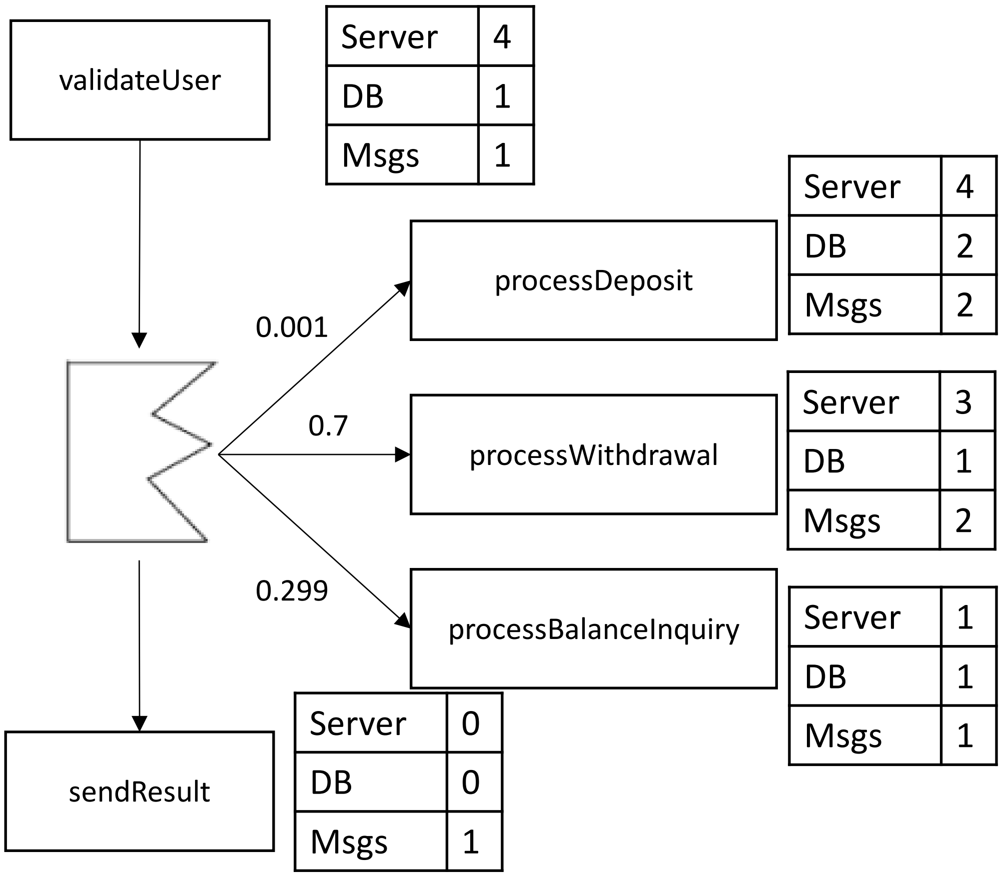


Execution graph



Exercise: processTransaction scenario

Estimate the best, worst, and average execution time



Ressources	CPU	Disk	Network
Quantity	1	1	1
Service unit	KInstr.	I/O	Msgs
Server	800	0	0
DB	200	2	1
Msgs	10	2	1
Service time	0.000015	0.005	0.001

Queuing Network Models
(System Execution Models)

Software Execution Models vs. Queuing Network Models

- **Software execution models**
  - provide a static analysis of the mean, best-and worst-case response times for software
  - characterize the resource requirements of the proposed software alone, in the absence of other workloads or multiple users
- **Queuing network models (QNM)**
  - characterize the software's performance in the presence of dynamic factors, such as other work loads or multiple users
  - aim to solve the contention for resources

If the software execution model indicates that there are no problems, then you are ready to construct and solve the queuing networks to account for contention efforts

Benefits of QNM

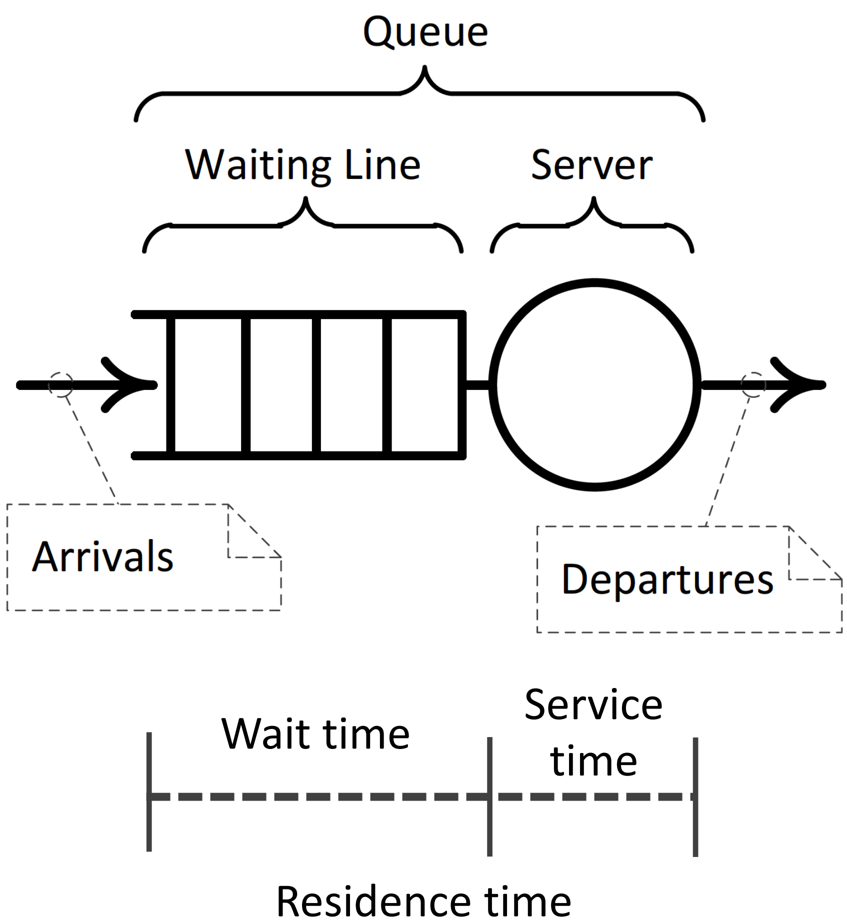
- More precise metrics that account for resource contention
- Sensitivity of performance metrics to variations in workload composition
- Scalability of the hardware and software to meet future demands
- Effect of new software on service level objectives of other systems
- Identification of bottleneck resources
- Comparative data on performance improvement options

Sources of Contention for Resources

- Multiple users of an application or transaction executing at one time, e.g. several ATM customers do a withdrawal simultaneously
- Multiple applications or systems executing on the same hardware resources at one time
- The application under consideration can have separate concurrent processes
- The application may be multi-threaded to handle concurrent requests for different external processes

QNM Basics: Queues

- The basic component of queuing networks is a ***queue***, also referred to as a ***service station*** or ***service center***.
- A queue consists of a waiting line and a server (e.g., CPU, disk, network), which serves incoming requests (a queue can have multiple servers)



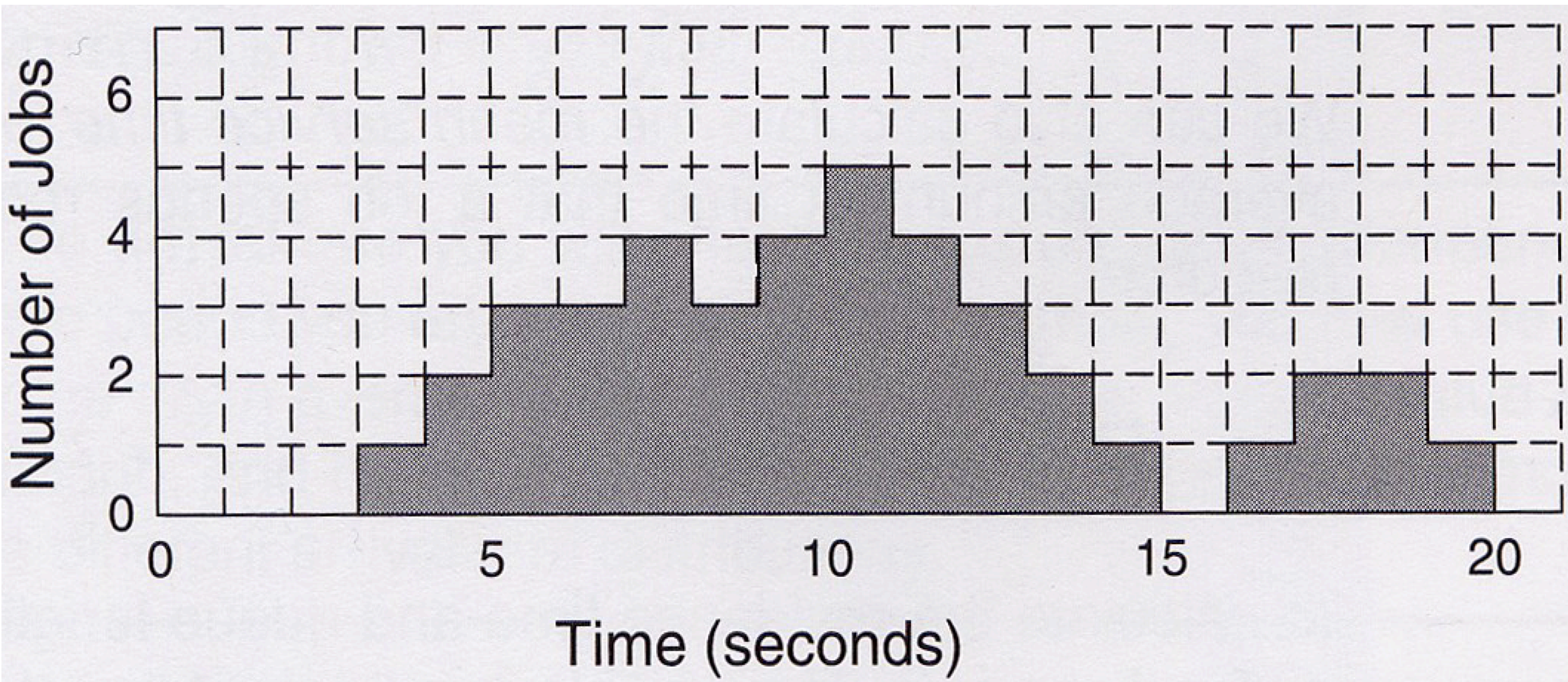
Performance Metrics

- Performance metrics of interest for each server are
  - ***Residence time***, RT: the average time jobs spend in the server, in service and waiting
  - ***Utilization***, U: the average percentage of the time the server is busy
  - ***Throughput***, X: the average rate at which jobs complete service
  - ***Queue length***, N: the average numbers of jobs at the server (receiving service and waiting)

Performance Metrics (con't)

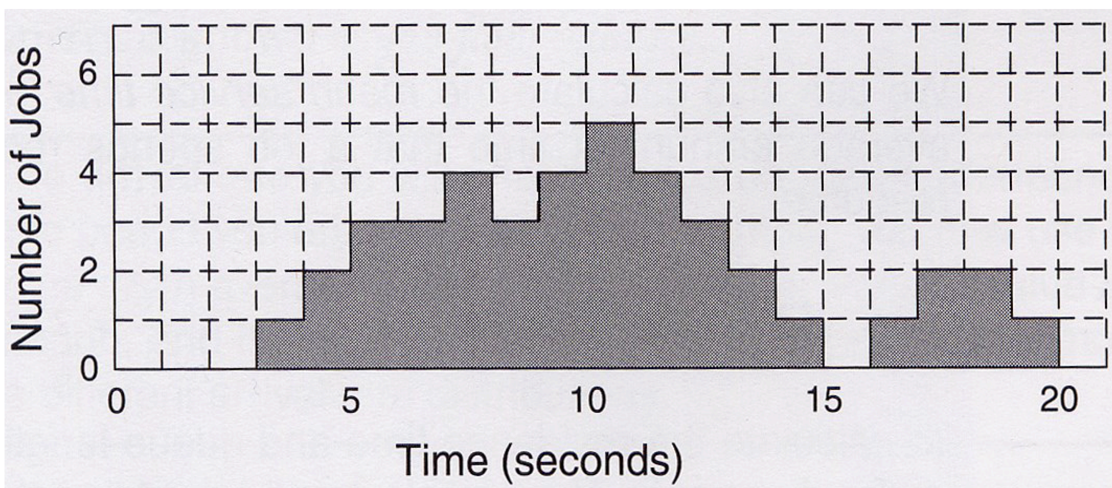
- The value of these metrics depend on
  - The number of jobs
  - The amount of service they need
  - The time required for the server to process individual jobs
  - The policy used to select the next job from the queue (e.g., the first-come-first-served or priority scheduling)

Execution Profile



Execution Profile (con't)

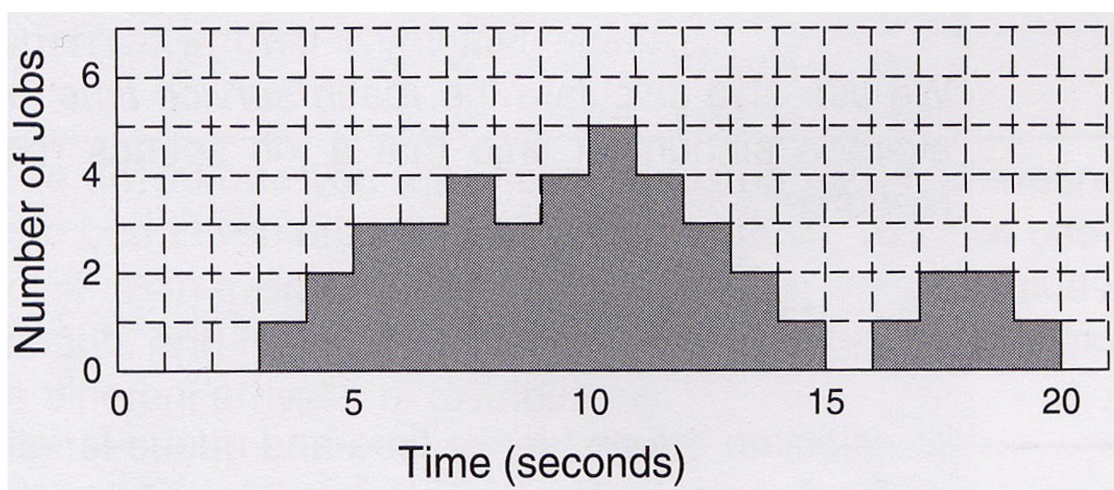
- From the execution profile, we obtain the following data:
  - Measurement period, T — 20 sec
  - Number of arrivals, A — 8 jobs
  - Number of completions, C — 8 jobs
  - Busy time, B — 16 sec



Calculation of Performance Metrics

T: Total Period
C: Completed
B: Busy Time

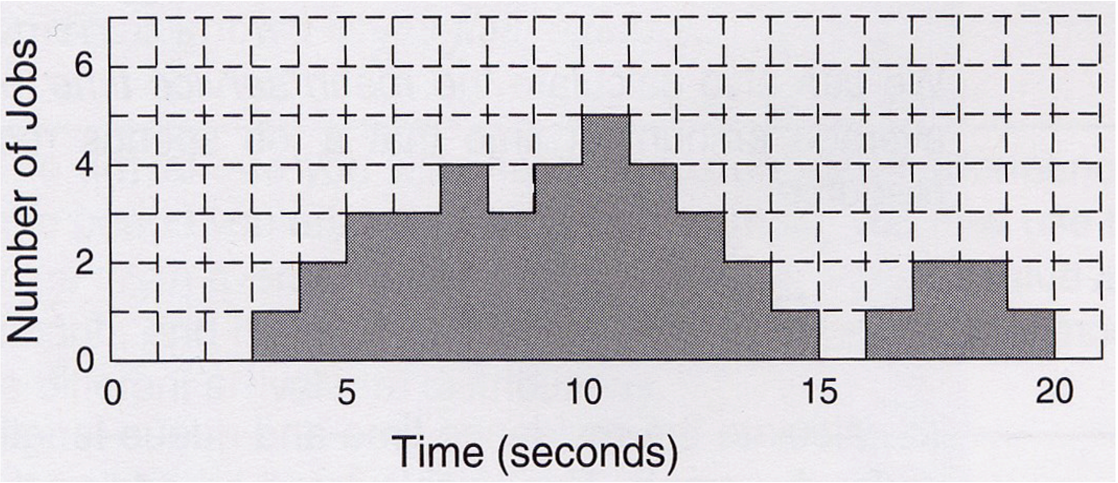
- Utilization, $U = B/T$
- Throughput, $X = C/T$
- Mean service time, $S = B/C$
- Area under graph, $W = \sum_{time}(\# \; of \; jobs)$
- Residence time, $RT = W/C$
- Queue length, $N = W/T$



Calculation of Performance Metrics

T: Total Period = 20
C: Completed = 8
B: Busy Time = 16

- Utilization, $U = B/T = 0.8$
- Throughput, $X = C/T = 0.4$ jobs/sec
- Mean service time, $S = B/C = 2$ sec
- Area under graph, $W = \sum_{time}(\#\ of\ jobs) = 41$ jobs
- Residence time, $RT = W/C = 5.125$ sec
- Queue length, $N = W/T = 2.05$ jobs



Solving the Queueing Model

- Use similar calculations, based on predicted ***workload intensity*** and ***service requirements***
- ***Workload intensity*** is a measure of the number of requests made by a workload in a given time interval
- ***Service requirements*** are the amount of time that the workload requires from each of the devices in the processing facility

Solving the Queueing Model (con't)

- Assume that the system is fast enough to handle the arrivals, and thus ***the completion rate or throughput equals the arrival rate***
- This property is called ***jobs-flow balance***

Example

- Workload intensity
  - Arrival rate, $\lambda = 0.4$ jobs per sec
- Service requirements
  - Mean service time, $S = 2$ sec

We then calculate the following average values:

- Throughput, $X = \lambda$
- Utilization, $U = XS$ (Utilization Law)
- Residence time, $RT = \frac{S}{1-U}$
- Queue length, $N = X * RT$ (Little Law)

Exercise: Use of Utilization Law

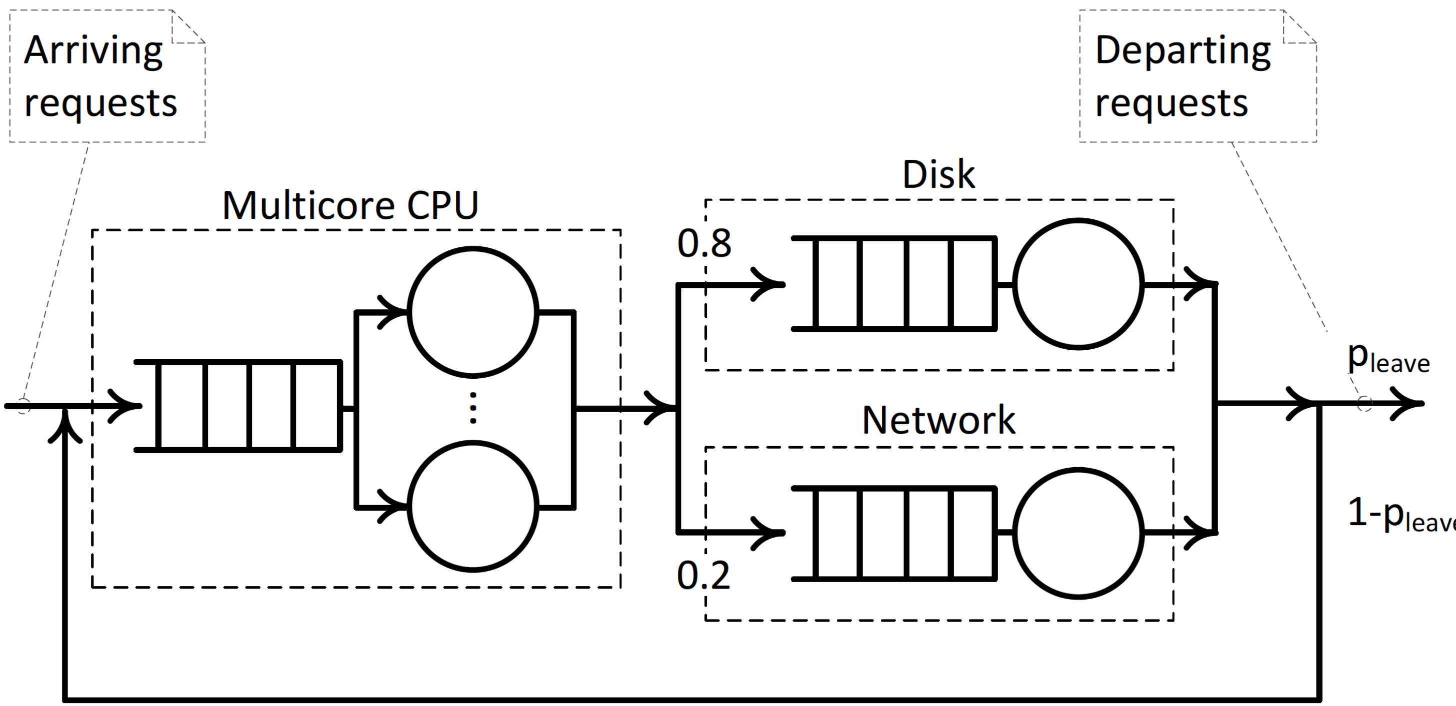
- A network segment transmits 1,000 packets/sec. Each packet has an average transmission time equal to 0.15 msec. What is the utilization of LAN segment?

Exercise: Use of Little Law

- An NFS server was monitored during 30 minutes and the number of I/O operations performed during the period was found to be 10,800. The average number of active NFS requests was found to be three. What was the average response time per NFS request at the server?

Queuing Network Models

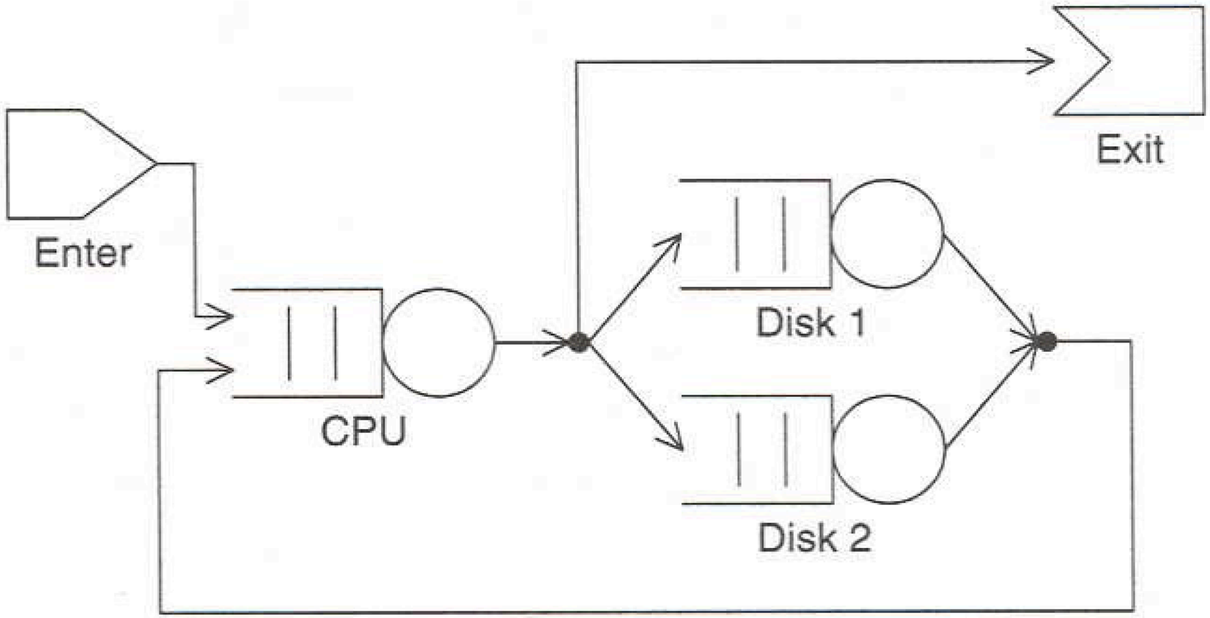
- A queueing network (QN) consists of two or more queues (service stations) that are connected together and serve requests sent by clients.
- The routing of requests in the queueing network is specified by a probability matrix.



Types of QNM

- **Open models**
  - The requests come from a source that is external of the queueing network and leave the network after service completion
- **Closed models**
  - No external source of requests and no departing requests (the population of requests in the queueing network remains constant)
- **Mixed models**
  - Open for some workload classes and closed for others

Open QNM



Open QNM (con't)

- Open QNM is appropriate for systems with external arrivals and departures, such as ATM
- For an open QNM, specify the ***workload intensity*** and ***service requirements***
- The workload is the ***arrival rate*** that rate at which jobs arrive for service
- The service requirements are ***the number of visits*** for each device, and ***the average service time per visit***, or ***the total demand*** for that device

Open QNM Computation

- We specify the following parameters:

  λ System arrival rate

  V_i Number of visits to device

  S_i Mean service time at device

- System throughput, X_0 $X_0 = \lambda$
- Throughput of device i, X_i $X_i = X_0 \times V_i$

  Utilization of device i, U_i, $U_i = X_i \times S_i$
- Residence time per visit at device i, RT_i

$$RT_i = \frac{S_i}{1 - U_i}$$

Open QNM Computation

- Queue length for device i, N_i

$$N_i = X_i \times RT_i$$

- System queue length, N

$$N = \sum_i N_i$$

- System response time, RT

$$RT = \frac{N}{X_0}$$

Example: Open QNM Solution

- Sample parameters

System arrival rate, $\lambda = 5$ jobs per second

Number of visits, V		**Mean service time, S**	
CPU	5	CPU	0.01
Disk1	3	Disk1	0.03
Disk2	1	Disk2	0.02

Example: Open QNM Solution

Metrics	CPU	Disk1	Disk2
1. **X**, throughput			
2. **S**, mean service time			
3. **U**, utilization			
4. **RT**, residence time			
5. **N**, queue length			
Total jobs in system =			
System response time =			

Example: Open QNM Solution

Metrics	CPU	Disk1	Disk2
1. **X**, throughput	25	15	5
2. **S**, mean service time	0.01	0.03	0.02
3. **U**, utilization	0.25	0.45	0.10
4. **RT**, residence time	0.013	0.055	0.022
5. **N**, queue length	0.325	0.825	0.111
Total jobs in system = 0.325 + 0.825 + 0.111 = 1.26			
System response time = 1.26/5 = 0.252 sec			

Exercise: what if the arrival rate doubles?

- Sample parameters

System arrival rate, λ = ~~5~~ 10 jobs per second

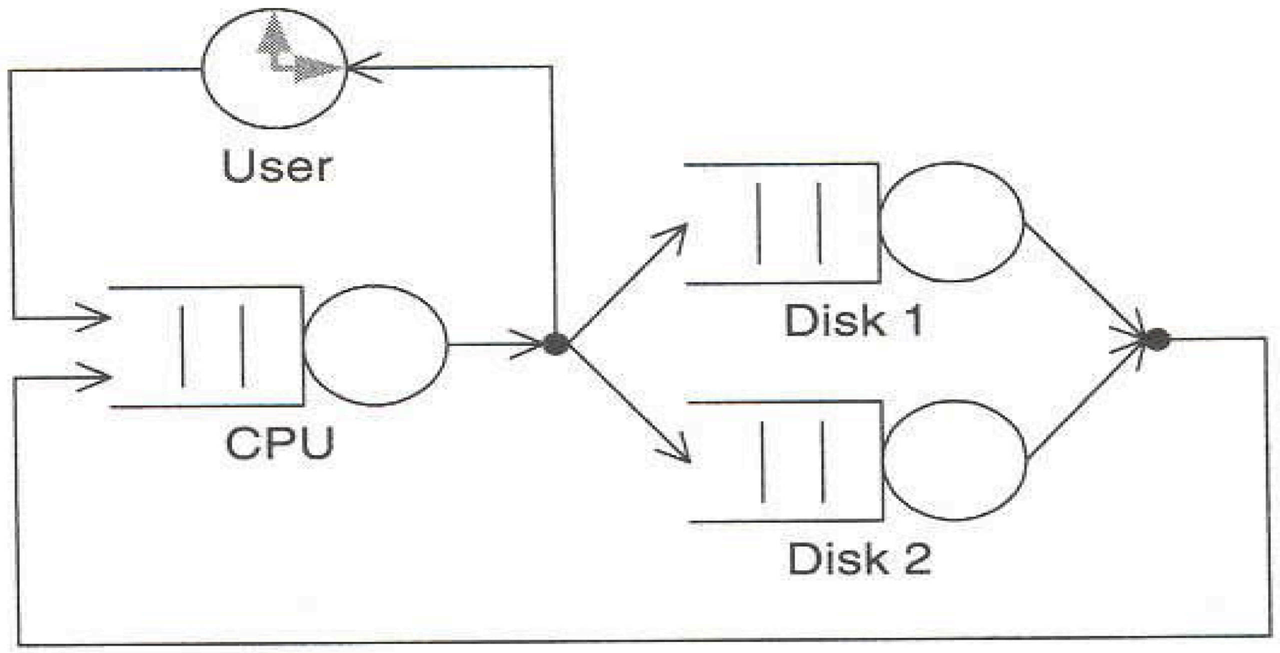
Number of visits, V		Mean service time, S	
CPU	5	CPU	0.01
Disk1	3	Disk1	0.03
Disk2	1	Disk2	0.02

Exercise: what if the arrival rate doubles?

Metrics	CPU	Disk1	Disk2
1. **X**, throughput			
2. **S**, mean service time			
3. **U**, utilization			
4. **RT**, residence time			
5. **N**, queue length			
Total jobs in system =			
System response time =			

Closed QNM

- Closed QNM has no external arrivals or departures
- A fixed number of jobs keep circulating among queues



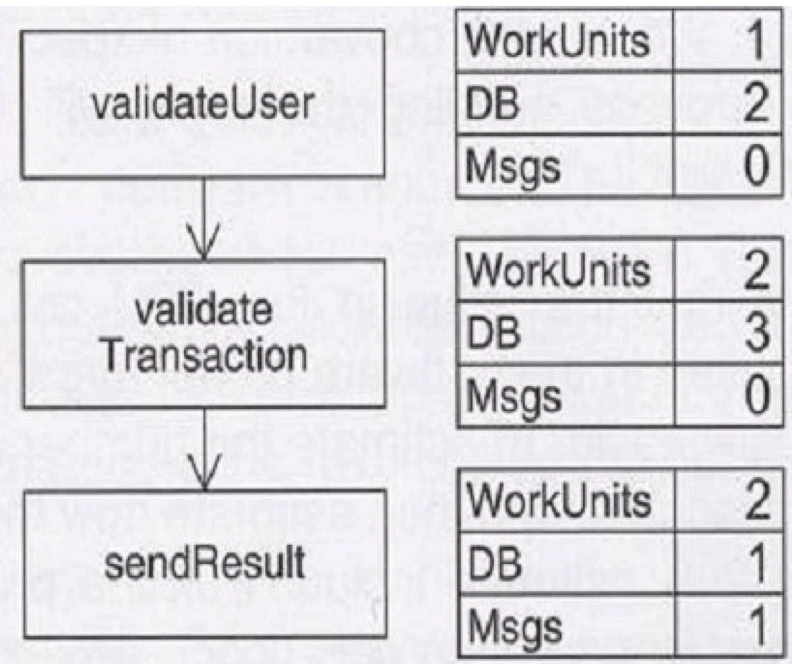
Solving Closed QNM

- This model needs
  - ***The number of users*** (or the number of simultaneous jobs)
  - ***The think time***, i.e., the average delay between the receipt of a response and the submission of the next
  - Number of visits
  - Service time
  - Total demand for each device

Deriving System Model Parameters from Software Model Results

- ***Step 1:*** use queue-servers to represent the key computer resources or devices that you specified in the software execution model and add the connections between queues to complete a model topology
- ***Step 2:*** decide whether the system is best modeled as an open or closed QNM
- ***Step 3***: determine the workload intensities for each scenario
- ***Step 4:*** specify the service requirements

Example: ATM authorizeTransaction Scenario

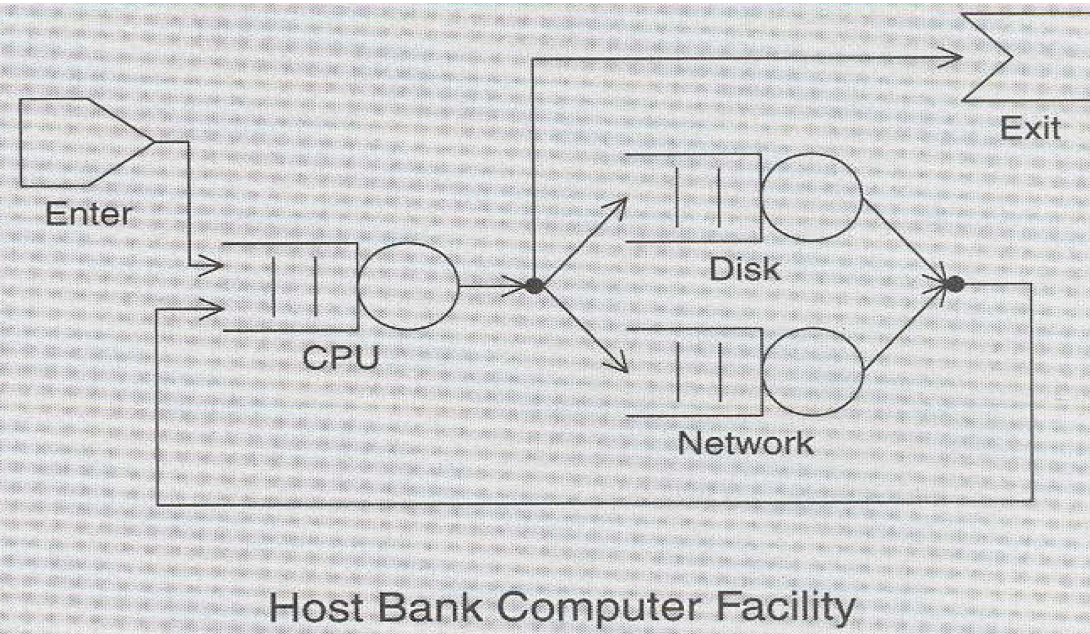


Software Model for authorizeTransaction

Table 4-1: Processing Overhead

Device	CPU	Disk	Network
Quantity	1	1	1
Service Unit	KInstr.	Phys. I/O	Msgs.
WorkUnit	20	0	0
DB	500	2	0
Msgs	10	2	1
Service time	0.00001	0.02	0.01

Example: ATM authorizeTransaction Scenario



Processing step	CPU Kinstr.	Disk	Network Messages
validateUser	1,020	4	0
validateTransaction	1,540	6	0
sendResult	550	4	1
Total: authorizeTransaction	3,110	14	1

Example: ATM authorizeTransaction Scenario

Device	Visits, *V*	Device Service Time, *S*
CPU	all	.0311
Disk	14	.02
Network	1	.01

Modeling Hints

- Multiple users and workload (e.g., arrival rate, the number of users, and think time)
- Average vs. Peak Performance
  - Basis QNMs calculates average values
- Sensitivity: if a small change in one parameter causes a large change in the computed metrics, the model is sensitive to that quantity
- Scalability: improves response times for your anticipated future loads
- Bottlenecks: the bottleneck device is the one with the highest utilization

Machine learning based performance models

Drawbacks of Queuing Network models

- Require extensive knowledge of the software and system
- Cannot handle passive resources (resource that are required for processing but do no work themselves, e.g., memory)
- Only estimates average metrics; cannot estimate minimum, maximum, variance, distributions, etc.
- Difficult to scale to large-scale software systems

- Can we treat of the software and system as a black-box?

Black-box (machine learning based) performance models

Black-box (machine learning based) performance models

X	$f(X)$	Y

Black-box (machine learning based) performance models

X	$f(X)$	Y

Given workloads (X) and measured performance metrics (Y), we can train a machine learning model (Y = f(X))

Workloads (independent variables)

Timestamp	Log line
00:00	Load data by Alice with size 32768 bytes
00:02	Read data by Alice with size 2048 bytes
00:03	Read data by Dan with size 1024 bytes
00:05	Read data by Alice with size16384 bytes
00:06	Write data by Alice with size 8192 bytes

In the six seconds:

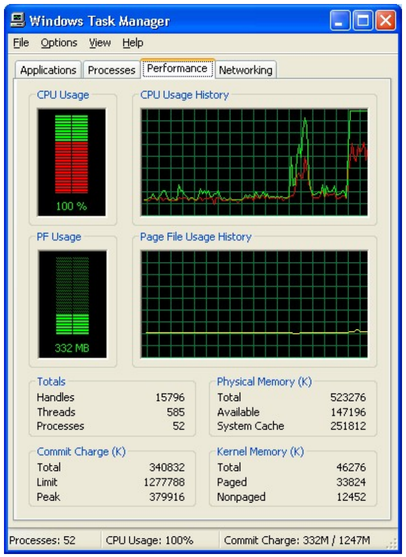
X(Load) = 1; X(Read) = 3; X(Write) = 1

Performance metrics (dependent variables)

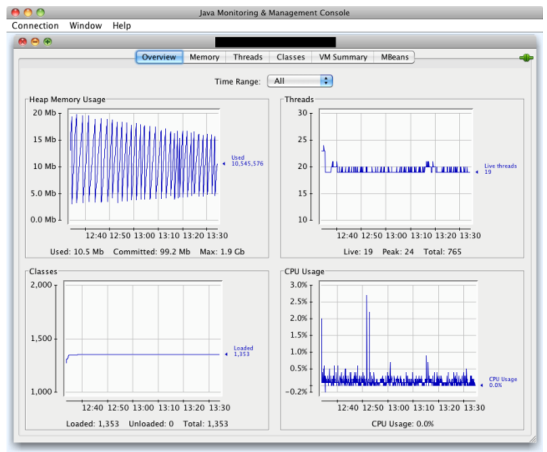
- Utilization (CPU, memory, disk, network)
- Response time
- Throughput
- Power consumption
- Battery
- …

How to collect performance metrics?

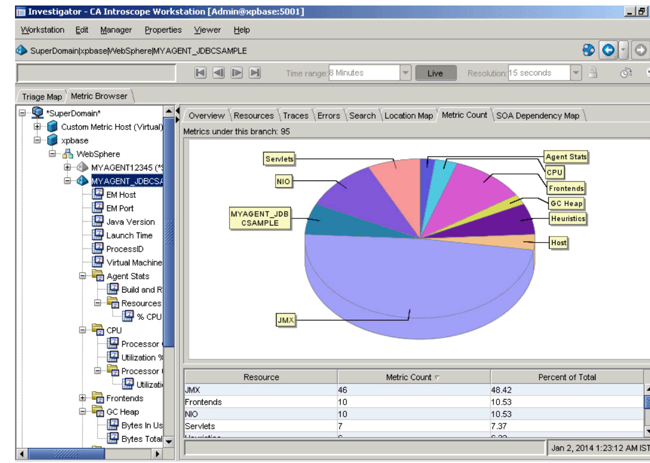
Monitoring



Task Manager

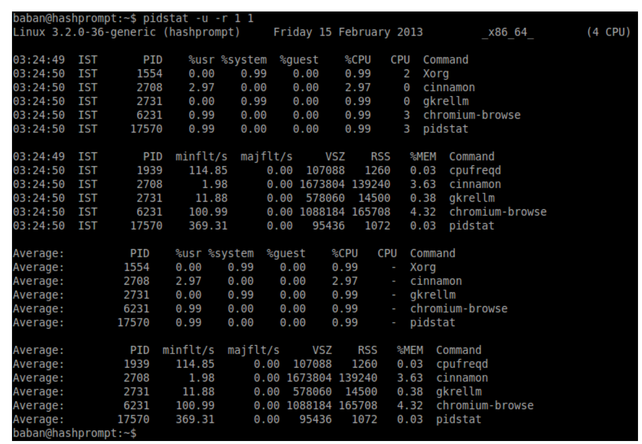


JConsole



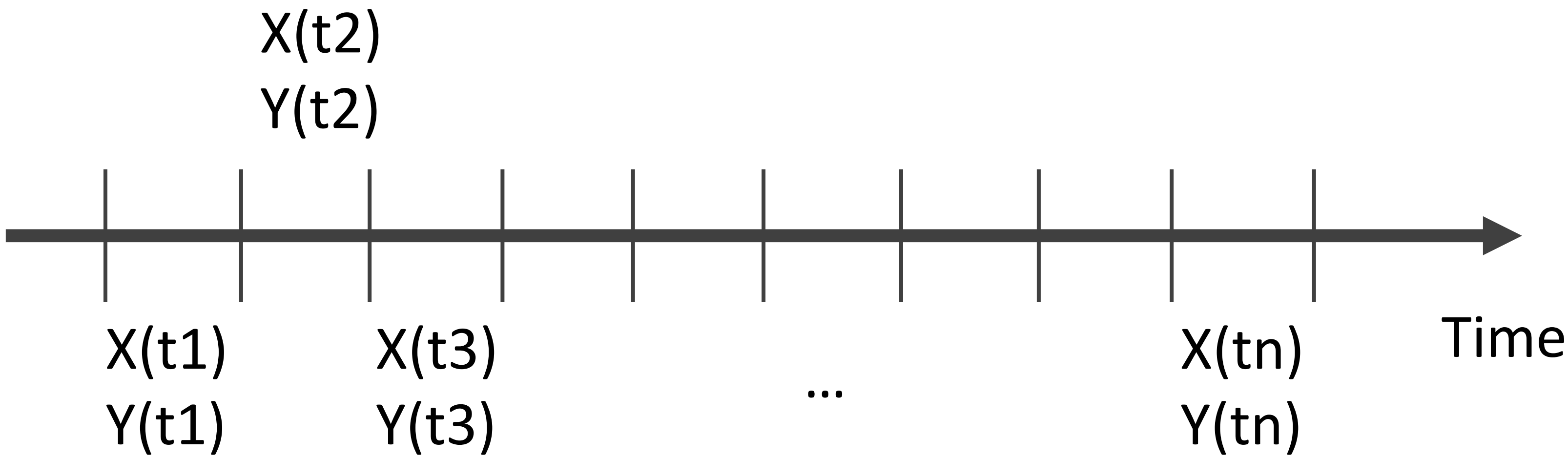
CA Willy

App Dynamics



pidstat

Aligning workloads and performance metrics by fixed time intervals



How to aggregate workload variables and performance metrics into fixed time intervals (e.g., every minute)?

- Count (for workloads)
- Mean/Min/Max/Median/Sum (for performance metrics)

Exercise: build a performance model for OpenMRS

- Given data:
  - Workload counts (extracted from logs) for every 30 seconds
  - Performance metrics (CPU utilization) averaged over every 30 seconds
- Build a machine-learning based performance model for OpenMRS
  - Given Python code in a Jupyter notebook
  - Using linear regression or random forest
- Download code and data:
  - https://drive.google.com/drive/folders/168q_9hAYmf2x6fmLPqjqlC48UTVBLl2y?usp=sharing