Code and Process Migration

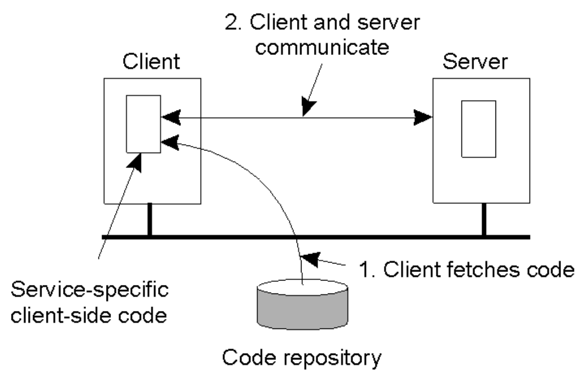
- Motivation
- How does migration occur?
- Resource migration
- Agent-based system
- Details of process migration

Motivation

- Key reasons: performance and flexibility
- Process migration (aka *strong mobility*)
  – Improved system-wide performance – better utilization of system-wide resources
  – Examples: Condor, DQS
- Code migration (aka *weak mobility)*
  – Shipment of server code to client – filling forms (reduce communication, no need to pre-link stubs with client)
  – Ship parts of client application to server instead of data from server to client (e.g., databases)
  – Improve parallelism – agent-based web searches

Motivation

- Flexibility
  - Dynamic configuration of distributed system
  - Clients don't need preinstalled software – download on demand



Migration models

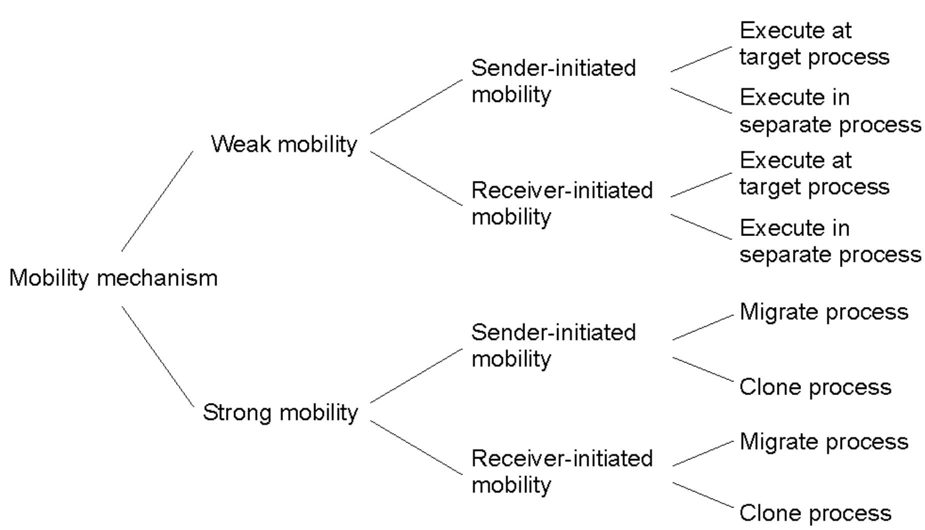
- Process = code seg + resource seg + execution seg
- Weak versus strong mobility
  - Weak => transferred program starts from initial state
- Sender-initiated versus receiver-initiated
- Sender-initiated (code is with sender)
  - Client sending a query to database server
  - Client should be pre-registered
- Receiver-initiated
  - Java applets
  - Receiver can be anonymous

Who executes migrated entity?

- Code migration:
  - Execute in a separate process
  - [Applets] Execute in target process
- Process migration
  - Remote cloning
  - Migrate the process

Models for Code Migration

-



Do Resources Migrate?

- Depends on resource to process binding
  - By identifier: specific web site, ftp server
  - By value: Java libraries
  - By type: printers, local devices
- Depends on type of "attachments"
  - Unattached to any node: data files
  - Fastened resources (can be moved only at high cost)
    - Database, web sites
  - Fixed resources
    - Local devices, communication end points

Resource Migration Actions

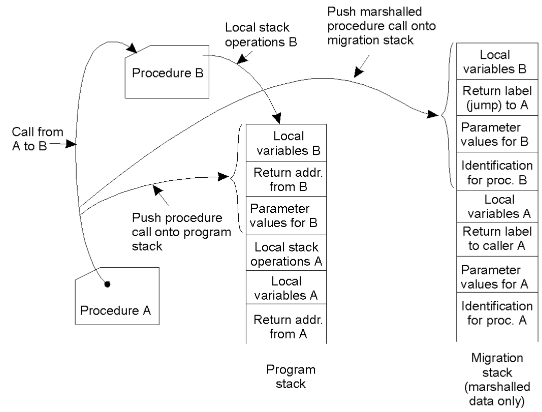
Resource-to machine binding

Process-to-resource binding		Unattached	Fastened	Fixed
	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV, GR)	GR (or CP)	GR
	By type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

- Actions to be taken with respect to the references to local resources when migrating code to another machine.
- GR: establish global system-wide reference
- MV: move the resources
- CP: copy the resource
- RB: rebind process to locally available resource

Migration in Heterogeneous Systems

- Systems can be heterogeneous (different architecture, OS)
  - Support only weak mobility: recompile code, no run time information
  - Strong mobility: recompile code segment, transfer execution segment [migration stack]
  - Virtual machines - interpret source (scripts) or intermediate code [Java]



Case study: Agents

- Software agents
  - Autonomous process capable of reacting to, and initiating changes in its environment, possibly in collaboration
  - More than a “process” – can act on its own
- Mobile agent
  - Capability to move between machines
  - Needs support for strong mobility
  - Example: D’Agents (aka Agent TCL)
    - Support for heterogeneous systems, uses interpreted languages

Case Study: ISOS

- Internet scale operating system
  – Harness compute cycles of thousands of PCs on the Internet
  – PCs owned by different individuals
  – Donate CPU cycles/storage when not in use (pool resouces)
  – Contact coordinator for work
  – Coodinator: partition large parallel app into small tasks
  – Assign compute/storage tasks to PCs
- Examples: Seti@home, P2P backups

Case study: Condor

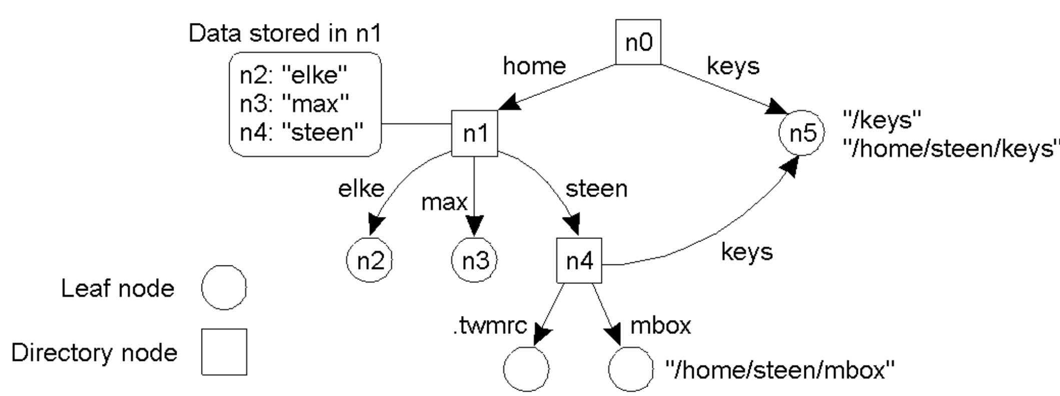
- Condor: use idle cycles on workstations in a LAN
- Used to run lareg batch jobs, long simulations
- Idle machines contact condor for work
- Condor assigns a waiting job
- User returns to workstation => suspend job, migrate
- Flexible job scheduling policies

New Topic: Naming

- Names are used to share resources, uniquely identify entities and refer to locations
- Need to map from name to the entity it refers to
  - E.g., Browser access to www.cnn.com
  - Use name resolution
- Differences in naming in distributed and non-distributed systems
  - Distributed systems: naming systems is itself distributed
- How to name mobile entities?

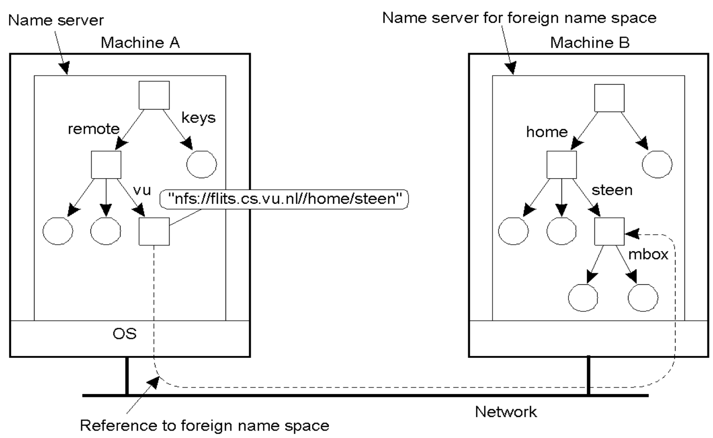
Example: File Names

- Hierarchical directory structure (DAG)
  - Each file name is a unique path in the DAG
  - Resolution of */home/steen/mbox* a traversal of the DAG
- File names are *human-friendly*



Resolving File Names across Machines

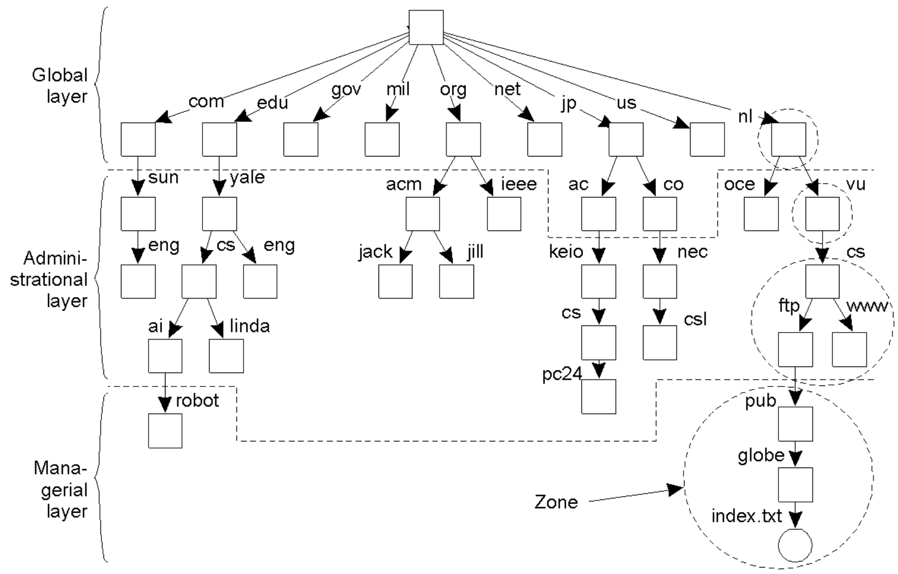
- Remote files are accessed using a node name, path name
- NFS mount protocol: map a remote node onto local DAG
  - Remote files are accessed using local names! *(location independence)*
  - OS maintains a mount table with the mappings



Name Space Distribution

- Naming in large distributed systems
  - System may be global in scope (e.g., Internet, WWW)
- Name space is organized hierarchically
  - Single root node (like naming files)
- Name space is distributed and has three logical layers
  - Global layer: highest level nodes (root and a few children)
    - Represent groups of organizations, rare changes
  - Administrational layer: nodes managed by a single organization
    - Typically one node per department, infrequent changes
  - Managerial layer: actual nodes
    - Frequent changes
  - Zone: part of the name space managed by a separate name server

Name Space Distribution Example



- An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

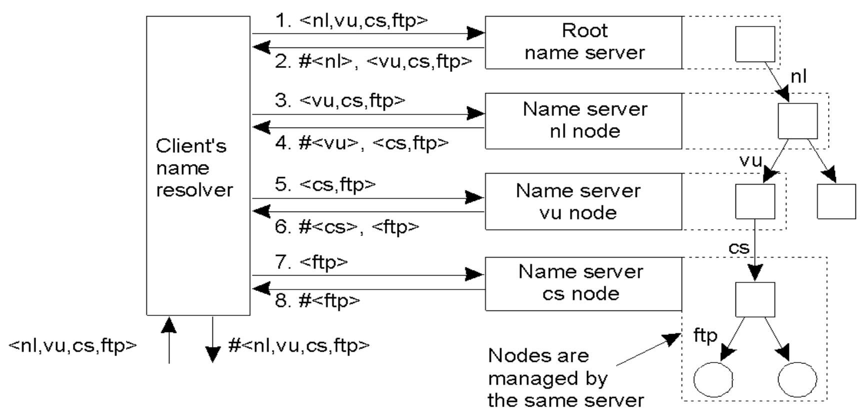
Name Space Distribution

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

- A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, as an administrational layer, and a managerial layer.
- The more stable a layer, the longer are the lookups valid (and can be cached longer)

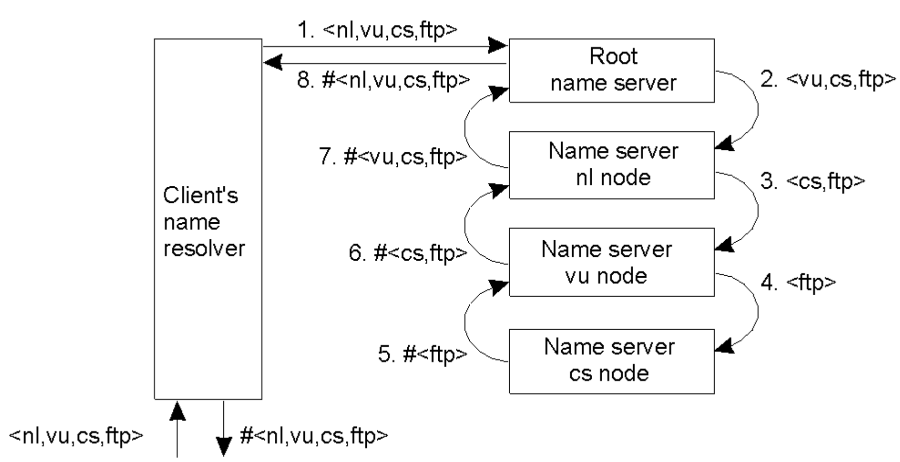
Implementing Name Resolution

- Iterative name resolution
  - Start with the root
  - Each layer resolves as much as it can and returns address of next name server



Recursive Name Resolution

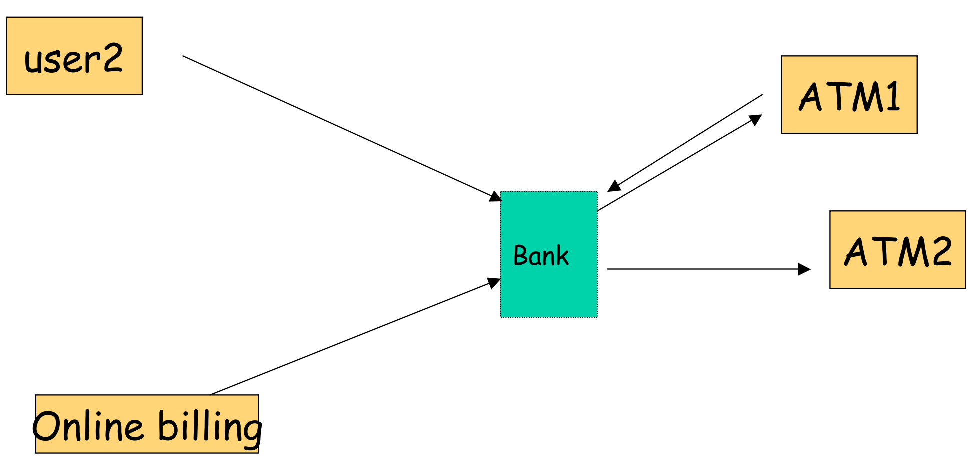
- Recursive name resolution
  - Start at the root
  - Each layer resolves as much as it can and hands the rest to the next layer



Project 1

- Illustrate distributed systems principles using an online bank
- Bank server: account information for various customers
- ATMs and online banking
  - Used to withdraw and deposit money
  - Pay bills, cash withdrawals

Online Bank



Project 1 details

- Bank server should be multi-threaded to service arbitrary number of online users and ATMs
  - Bank sever, users, ATMs can reside on different machines
- Sever should employ synchronization
  - Server may process data from multiple entities accessing the same account
  - Example: deposit $100, add $1 interest, withdraw $50 for online bill payment etc.