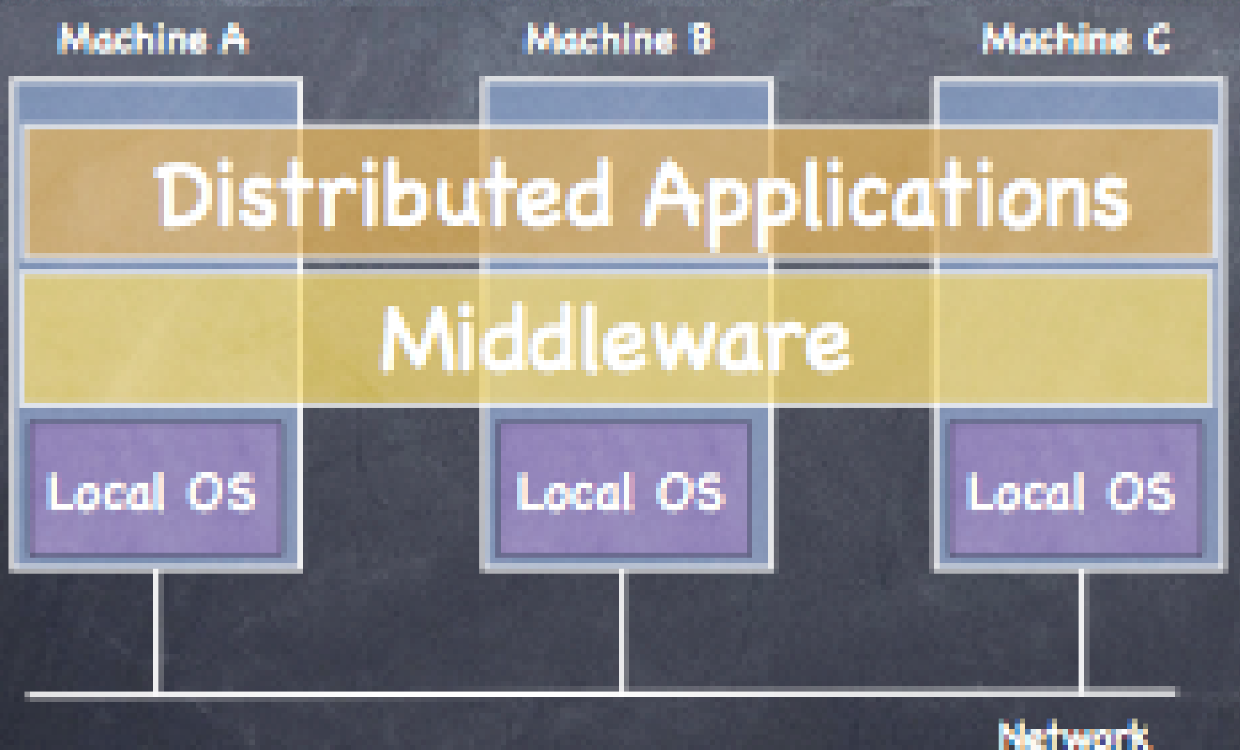
CS677
Distributed Systems

V. Arun
UMass Amherst

What is a distributed system?

A distributed system is software through which a collection of independent computers appears to its users as a single coherent system



Goals of a distributed system

- Connecting resources and users
- Transparency
- Openness
- Scalability

Transparency

Transparency	Description
Access	Hides difference in data representation and invocation mechanisms
Location	Hides where an object resides
Migration	Hides from an object that object's location
Relocation	Hides from a client the change of location of an object to which the client is bound
Replication	Hides that an object may be replicated with replicas at different locations
Concurrency	Hides coordination of activities between objects
Failure	Hides the failure and recovery of an object
Persistence	Hides whether a resource is in memory or on disk

Openness

- Confirm to well-defined interfaces
- Easily interact with other open systems
- Achieve independence in heterogeneity wrt
  - Hardware
  - Platform
  - Languages
- Support different app/user-specific policies
- ideally provide only mechanism

Scalability

- Size scalability
  - Number of users and processes
- Geographical scalability
  - Maximum distance between nodes
- Administrative scalability
  - Number of administrative domains

Scaling

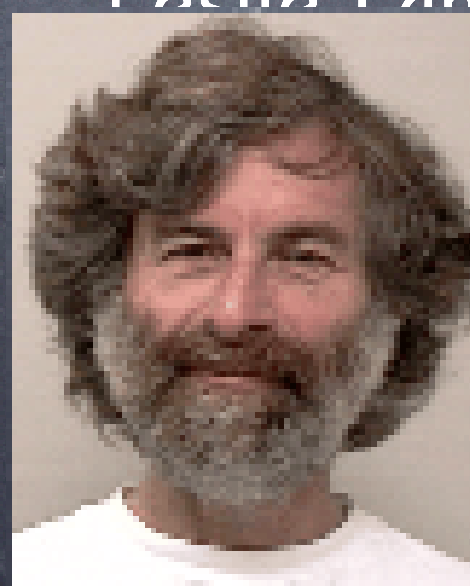
- Distribute
  - Partition data and computation across multiple machines
    - Java applets, DNS, WWW
- Repl
  - M ines
    - d DB
- Cache
  - Allow client processes to access local copies
    - Web caches, file caching

Consistency

What is a distributed system?

"A distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable"

Leslie Lamport



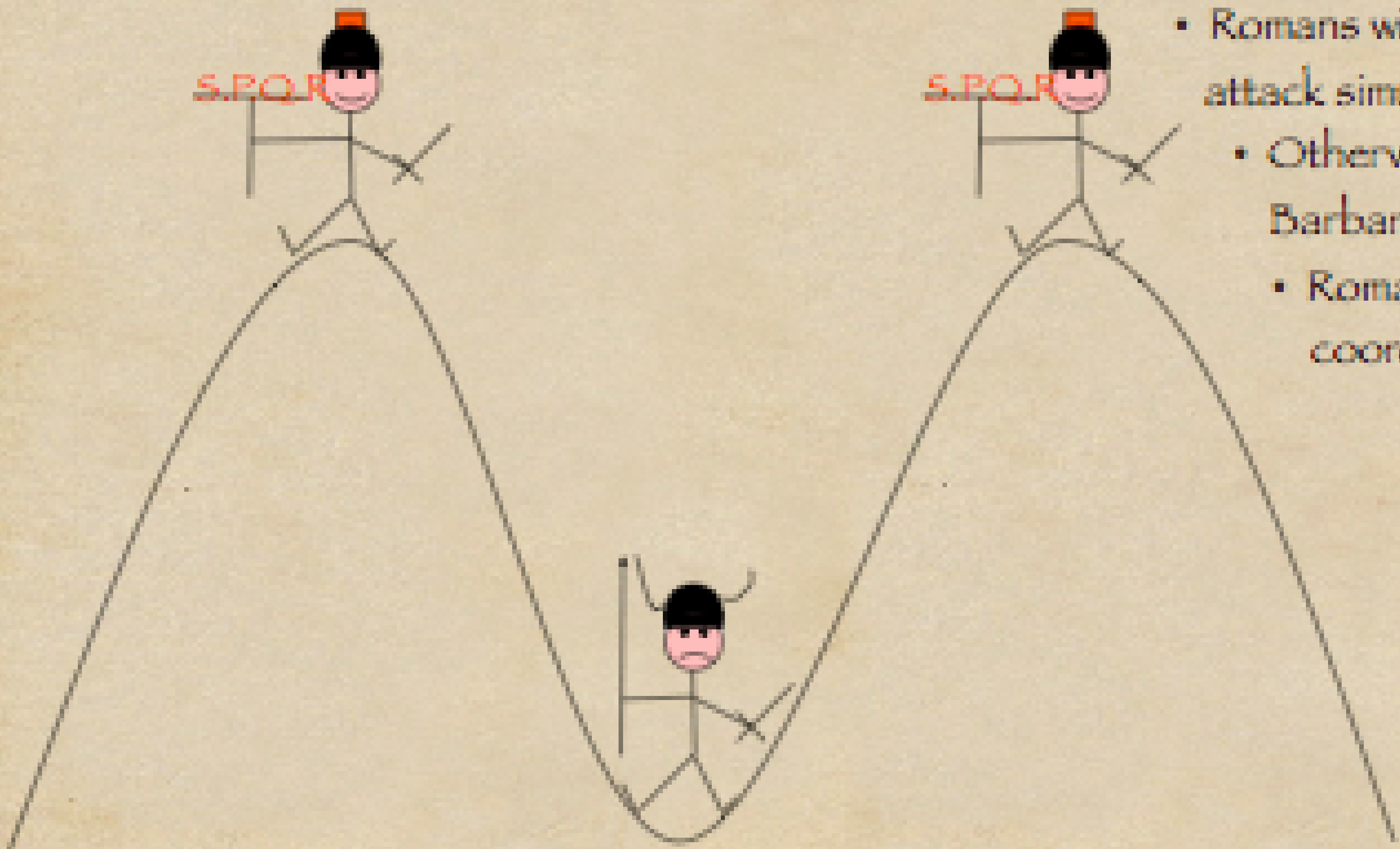
A first course in Distributed Systems...

- Two basic approaches
  - Cover many interesting systems, and distill from them fundamental problems
  - Focus on deep understanding of the fundamental principles, and see them instantiated in a few systems

Some intriguing questions

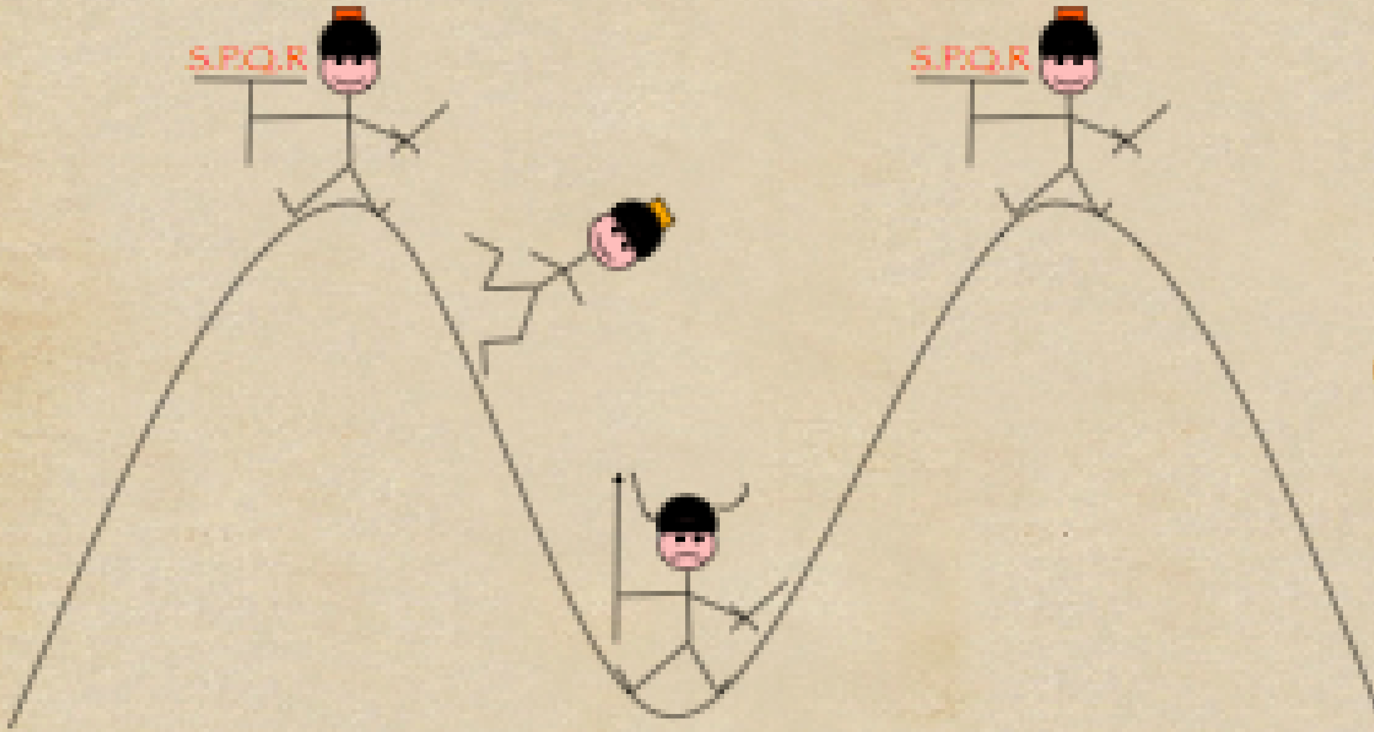
- How do we talk about a distributed execution?
- Can we draw global conclusions from local information?
- Can we coordinate operations without relying on synchrony?
- For the problems we know how to solve, how do we characterize the "goodness" of the solution?
- Are there real problems that simply cannot be solved?
- What are useful notions of consistency, and how do we maintain them?
- What if part of the system is down? Can we still do useful work? What if instead part of the system becomes "possessed" and starts behaving arbitrarily – are all bets off?

Two Generals' Problem



- Romans win if attack simultaneously
  - Otherwise, Barbarians win
    - Romans must coordinate attack

Two Generals' Problem



Problem:

Save Western Civilization

(i.e. design a protocol that ensures Romans always attack simultaneously)

- only communication is by messenger
- messengers must sneak through the valley
- they don't always make it

Two General's Problem

Claim: There is no non-trivial protocol that simultaneously guarantees that the Romans will always attack simultaneously

Two Generals Problem

Claim: There is no non-trivial protocol that simultaneously guarantees that the Romans will always attack simultaneously

Proof: By contradiction, consider a protocol that solves the Two Generals problem using the least number of messages. Let that number be n. Consider the n'th message m_{last}

The state of sender of m_{last} cannot depend on m_{last} receipt

The state of receiver of m_{last} cannot depend on m_{last} receipt

So both sender and receiver would come to the same conclusion even without sending m_{last}

We now have a new solution requiring only n-1 messages!

Two Generals Problem

Claim: There is no non-trivial protocol that simultaneously guarantees that the Romans will always attack simultaneously

Proof: By contradiction, consider a protocol that solves the Two Generals problem using the least number of messages. Let that number be n. Consider the n'th message m_{last}

The state of sender of m_{last} cannot depend on m_{last} receipt

The state of receiver of m_{last} cannot depend on m_{last} receipt

So both sender and receiver would come to the same conclusion even without sending m_{last}

We now have a new solution requiring only n-1 messages!

Conclusion: A solution requires reliable delivery

Global Predicate Detection and Event Ordering

Our Problem

To compute predicates
over the state of
a distributed application

Model

- Message Passing
- No node failures
- Two possible timing assumptions:
  - Synchronous system
  - Asynchronous system
    - No upper bound on message delivery time
    - No bound on relative process speeds
    - No centralized clock

Asynchronous systems

- Weakest possible assumptions
- Weak assumptions = less vulnerabilities
- Asynchronous != slow
- “Interesting” model w.r.t. failures