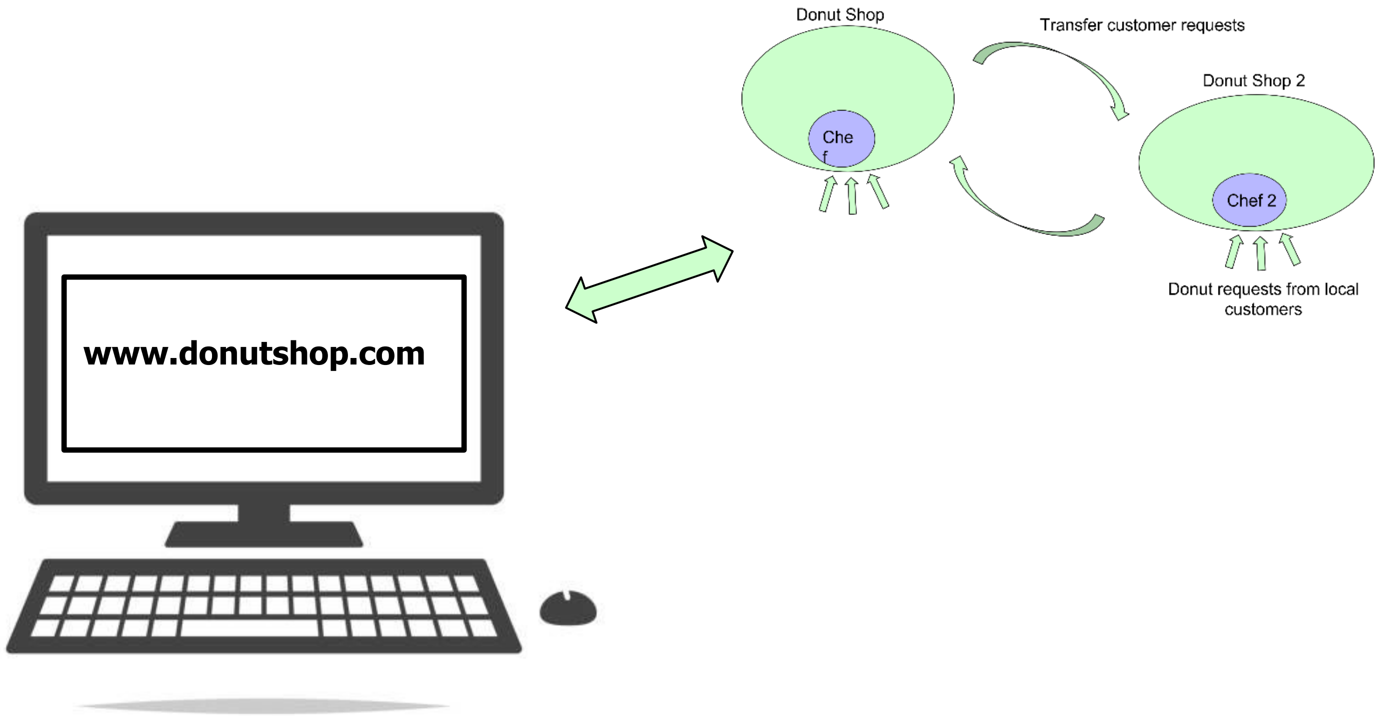
CEG 7370:

Distributed Computing

Salient Features of Distributed Systems

Intuition



www.donutshop.com
Donut Shop
Che
f
Transfer customer requests
Donut Shop 2
Chef 2
Donut requests from local customers

What is a Distributed System?

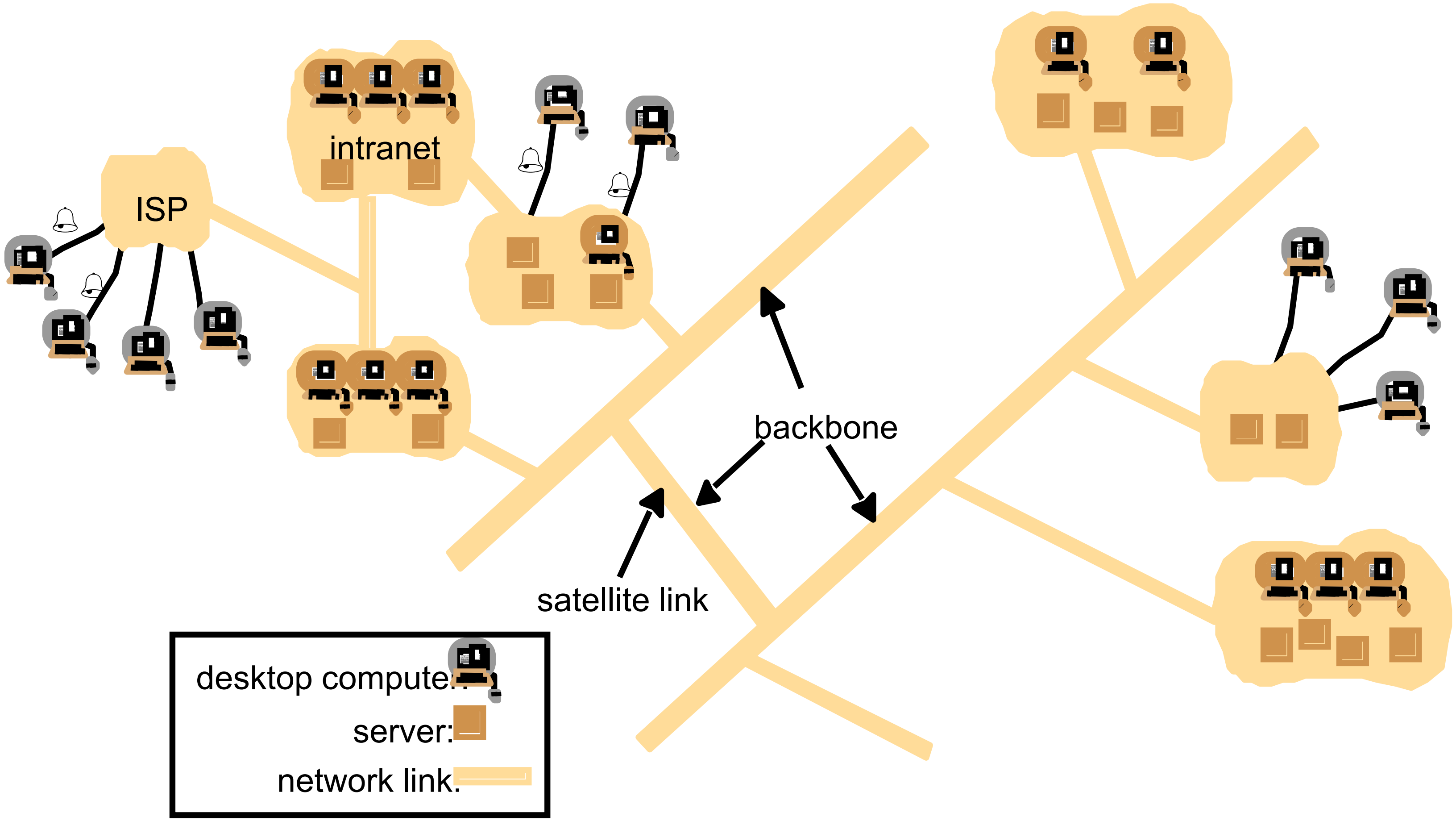
A Distributed System is:

a collection of independent computers that appear to the users as a single computer

Andrew Tannenbaum

- **Many components – Hardware and software**
- **Connected via a network**
- **Coordinate to accomplish a task or "Provide service" – "Web search", "Gaming", "smart home"**

Distributed System Example -- the Internet



Features of the Internet

- A vast interconnected collection **of computer networks of many types**.
- **Intranets** – **subnetworks** operated by companies and organizations.
- **ISPs** – companies that provide modem links and other types of connections to users.
- Intranets are linked by **backbones** – **network links of large bandwidth,** such as satellite connections, fiber optic cables, and other high-bandwidth circuits.

In this Course: Computing Infrastructure for Applications

Infrastructure (in-class)	Applications (Course project)
1. Storage (DFS) 2. Commmunication(Computer Networks) 3. Computations (Middleware, remote procedure calls etc)	1. Team Project

What is the significance of Independent Systems working together?

- **Concurrency:**
  - I do my work on my computer, you do yours but we share resources (web pages, memory, storage, filesystem)
  - The capacity of system to handle shared resources is done through two ways: 1) Increase resources, and/or 2) Implement effective coordination of resource-sharing

- **Independent failure:**

- all systems fail. But ensuring that they fail independent of others is important to keep the whole system running

Why build Distributed Systems? – To gain high performance

- To achieve parallelism

- To tolerate faults in the system

- To solve problems that are span over a geographically larger area – banking

- To ensure security → selective isolation(Reveal only as much as needed)

Why are Distributed Systems hard?

- **Parallelism→** You have to deal with complexities involved in concurrent programming → for example: resource sharing (how do you decide which process gets the resource?)

- **Unpredictable and multiple points of Failure→** Partial failure: sometimes some computers may work in the network.

Full failure: sometimes all computers may not work or the network itself may fail.

- Highly **variable** bandwidth
- Possibly large and **variable** latency
- No global clock; no single global notion of the correct time

Characteristics of Distributed Systems

- Transparency
- Performance
- Fault – Tolerance
- Consistency

Transparency - The Importance of Abstractions in Distributed Systems

- In both the Storage and Computations section of Distributed Systems we are interested in building **abstractions that mask the distributed nature** of underlying infrastructure

- **These abstractions can be interfaces (for example APIs)** that give the "appearance" to the calling program that it is dealing with a single monolithic system – 1) Non-distributed 2) Fault-tolerant

- Examples of abstractions 1) Remote Procedure calls 2) gRPC (developed by google) 3) Threads (structure concurrent operations using locks thereby masking programmer view)

Transparency

- Concealment of the separation of components from users:
  - **Access transparency:** local and remote resources can be accessed using identical operations.
  - **Location transparency:** resources can be accessed without knowing their whereabouts.
  - **Concurrency transparency:** processes can operate concurrently using shared resources without interferences.
  - **Failure transparency:** faults can be concealed from users/applications.
  - **Mobility transparency:** resources/users can move within a system without affecting their operations.
  - **Performance transparency:** system can be reconfigured to improve the performance.
  - **Scaling transparency:** system can be expanded in scale without change to the applications.

Transparency - Examples

- **Distributed file system** allows access transparency and location transparency.
- **URLs are location transparent,** but are not mobility-transparent (someone's personal web page cannot move to a new place and still be accessed using the same URL).
- Message retransmission governed by TCP is a mechanism for providing failure transparency.
- Mobile phone is an example of mobility transparency.

Fault Tolerance

- Yes, solving-big problems with LOTS of computers comes at a cost!! (There is always something WRONG!!)
- **In DS: many points of failure** → cables, switches, infrastructure change
- **It is impractical to address** every such individual problem but we can effectively mask it by making the system tolerate faults
- **Definition:** How well can a system tolerate failure by masking it and thereby still provide the same robust performance

Two main concepts in Fault Tolerance – What it means to be fault-tolerant?

- **Availability(stronger):** 1) systems are designed to keep operating to withstand some failures. Example: replicated servers

  2) in other words: under certain set of failures these systems work

- **Recoverability(weaker):** If something goes wrong the systems will stop responding. It will wait for repairs and after that it will resume function like nothing went wrong.

  - Common features of recoverability is that such systems would store data on disk so that they would start operating by loading data from that last point of failure.

A word on Availability and Recoverability

- All available systems with sufficient failures stop working at **some point**
- At these points these systems **must be recoverable to resume** their functionality.
- So **good available systems must also be recoverable**

Two tools to solve these problems

- **Non-volatile storage –** management of non-volatile storage can be very slow to read and write

- so an important consideration of building high-performance fault-tolerant systems **is to find clever ways to “Not write to non-volatile” storage.**

WRIGHT STATE UNIVERSITY

Two tools to solve these problems

- **Replication:** 1) management of replicated copies of systems is **tricky.**

2) Replicated copies **can drift away** from one another

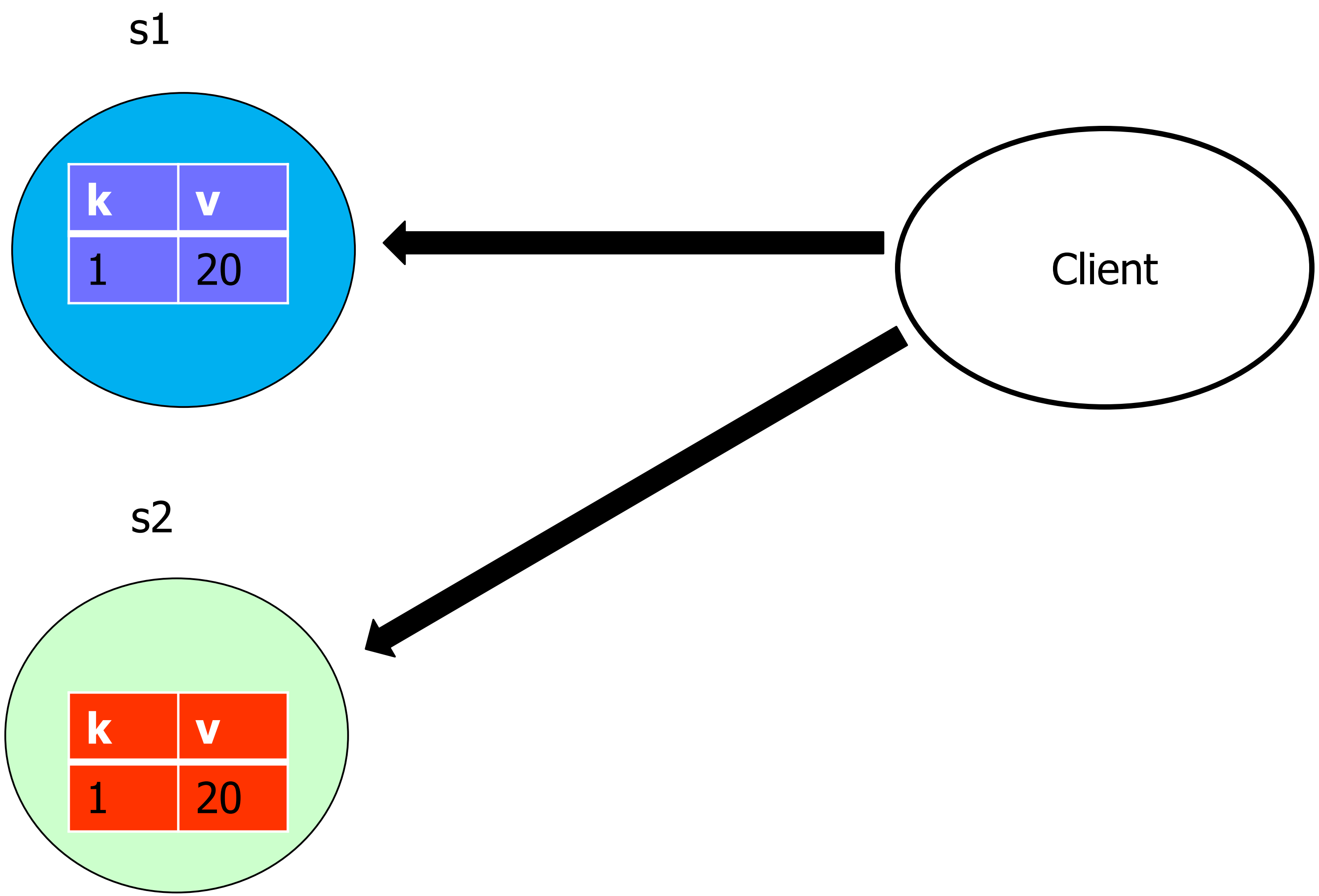
Consistency

- Consider a distributed storage server that stores (Key, value) pairs.
- Assume that this service supports two operations – 1) put(k, v) 2) get(k)→ v
- To gain clarity it is preferable to have a detailed description of **what** they do?
  - What it means ….to put(k,v)……what are the effects of such operation?
  - What it means……to get(k,v)……..what are its implications?
    - Usually a well-maintained api doc gives us this information

Consistency and Fault-tolerance

- In centralized monolithic designs there is little doubt or ambiguity pertaining to the semantics of **put and get**
- **In a distributes system, owing to the notion of fault-tolerance, there is a necessity for us to create multiple copies of data (replication)**
- These copies of data can be housed in different hardware systems that may be separated geographically by a large distances

Consistency and Fault-tolerance



Consistency - observations

- There is **serious issues of data inconsistency**

- **Possible solutions:** consider s1(refer previous slide) to be the top server……..

Implications of Inconsistency

- Deviation in replicated copies of computing systems…

- So NOW revisiting the semantics of put and get → we see that it has changed!

----→ How can we redefine put/get semantics?

Redefining semantics

- Possibility 1: get yields the value of the most recent "put" operation →

making an attempt to offer some **level of assurance or guarantee – Strong consistent systems**

- However this **is an expensive spec --→ why? – Communication overhead(Lot of chitchat)**

- **This problem is compounded by another important factor.........think independent failure and fault tolerance..**

Redefining semantics

- **Possibility 2:**
  - Donot guarantee that the value of get() yields the most recent put()
  - This reduces the communication overhead
  - Disadvantage: you may not get the most updated information

Performance

- An important aspect of performance is the idea of **scalable speed-up**
- **"Solve the problem in half the amount of time" –**

  - Two possible ways to achieve this:

  1) Increase the number of computers
  2) Hire personnel to optimize performance of existing resources

→ Which is more cheaper???

Performance , Consistency…….