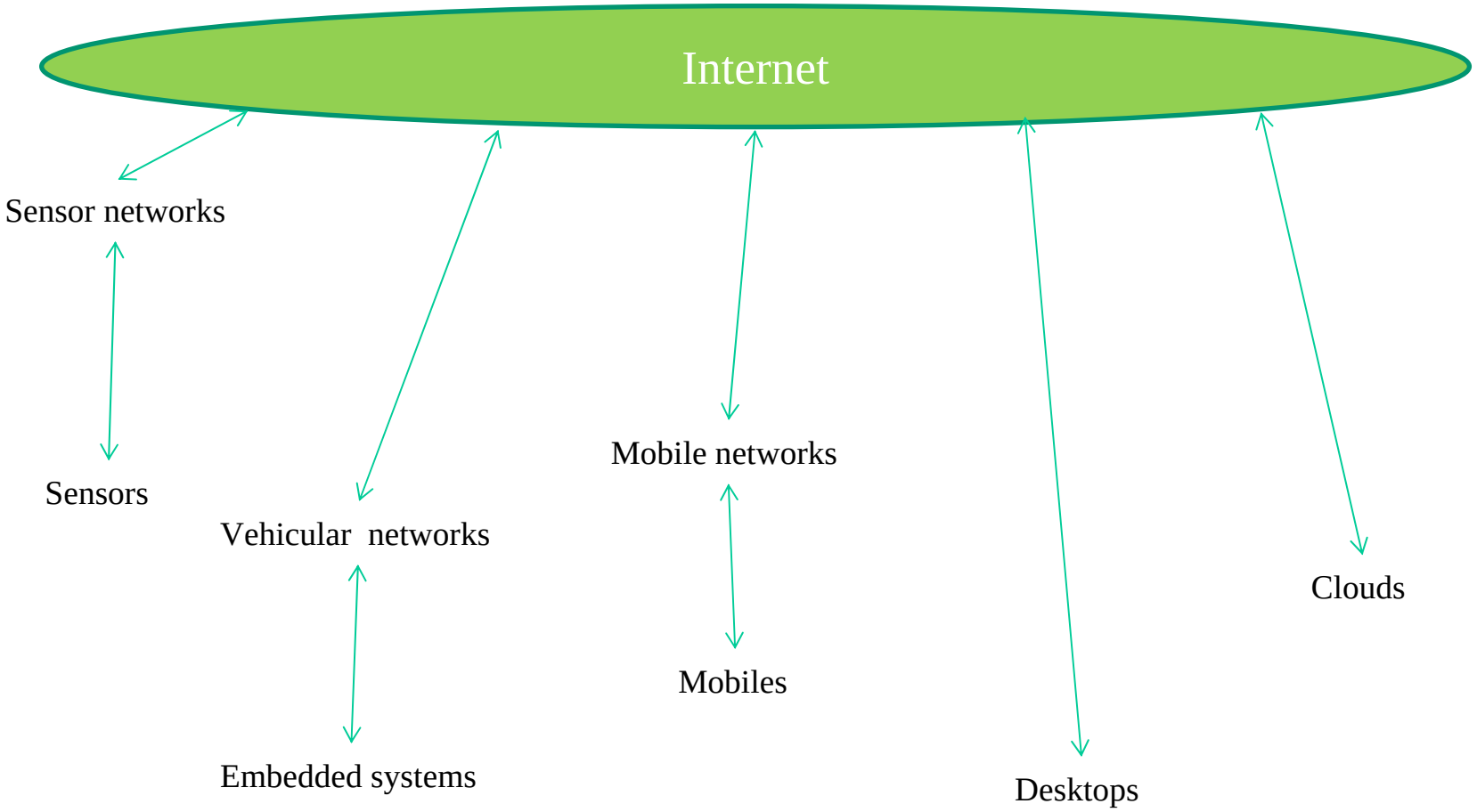
Lecture 18
CS 2506

Trends in operating systems

The new computing environment requires a new approach for the way OS services are provided.



New OS requirements

- The principal programming abstractions available today (processes, threads, files, sockets) do not adequately address the problems of managing locality, mobility, availability, scalability or fault tolerance.
- The solution can be a distributed OS that will enforce full location transparency: any code fragment might run anywhere, any data object might be hosted anywhere. The system will manage the locality, replication, and migration of computations and data.
- The system needs to be self-configuring, self-monitoring, and self-tuning, scalable and secure.

Millennium project at Microsoft Research

- **Goals** considering a user- or application-centred approach:
  - *Seamless distribution*. The system should determine where computations execute or data resides, moving them dynamically as necessary. Users should be able to use any computing device that is part of the distributed system as naturally and productively as they would use the machine on their desk at home or office.
  - *Worldwide scalability*. Logically there should be only *one system*, *although at any one time it may be partitioned into* many pieces. For example, disconnected or weakly-connected operations creates temporary network partitions.
  - *Fault-tolerance*. The system should transparently handle failures or removal of machines, network links, and other resources without loss of data or functionality. This should hold true for both the system itself and for its applications.

- *Self-tuning*. The system should be able to reason about its computations and resources, allocating, replicating, and moving computations and data to optimize its own performance, resource usage, and fault-tolerance.
- *Resource controls*. Both providers and consumers may explicitly manage the use of resources belonging to different trust domains. For instance, while some people might be content to allow their data and computations to use any resources available anywhere, some companies might choose, for instance, not to store or compute their year-end financial statement on their competitors machines.
- *Self-configuration*. New machines, network links, and resources should be automatically assimilated.
- *Security*. Although a single system image is presented, data and computations may be in many different trust domains, with different rights and capabilities available to different security principals. Like the Internet, the system should allow non-hierarchical trust domains with no central authority necessary.

Implementation principles

- *Aggressive abstraction*. The level of abstraction should be raised to the point that application programmers are freed from the mechanics of distributed programming and the constraints of physical computing components. This would allow them to focus on application rather than system aspects such as communication or fault tolerance. To the greatest extent possible, the system should handle difficult issues like data placement, resource location, fault-tolerance, and load-balancing.

- *Storage-irrelevance*. There should be no storage hierarchy. Once created, information should be accessible until it is no longer needed or referenced.

- *Location-irrelevance*. Objects should be allowed to reference each other and invoke operations without regard for their current location or replication state. The system should have a seamless appearance despite its underlying distributed nature.

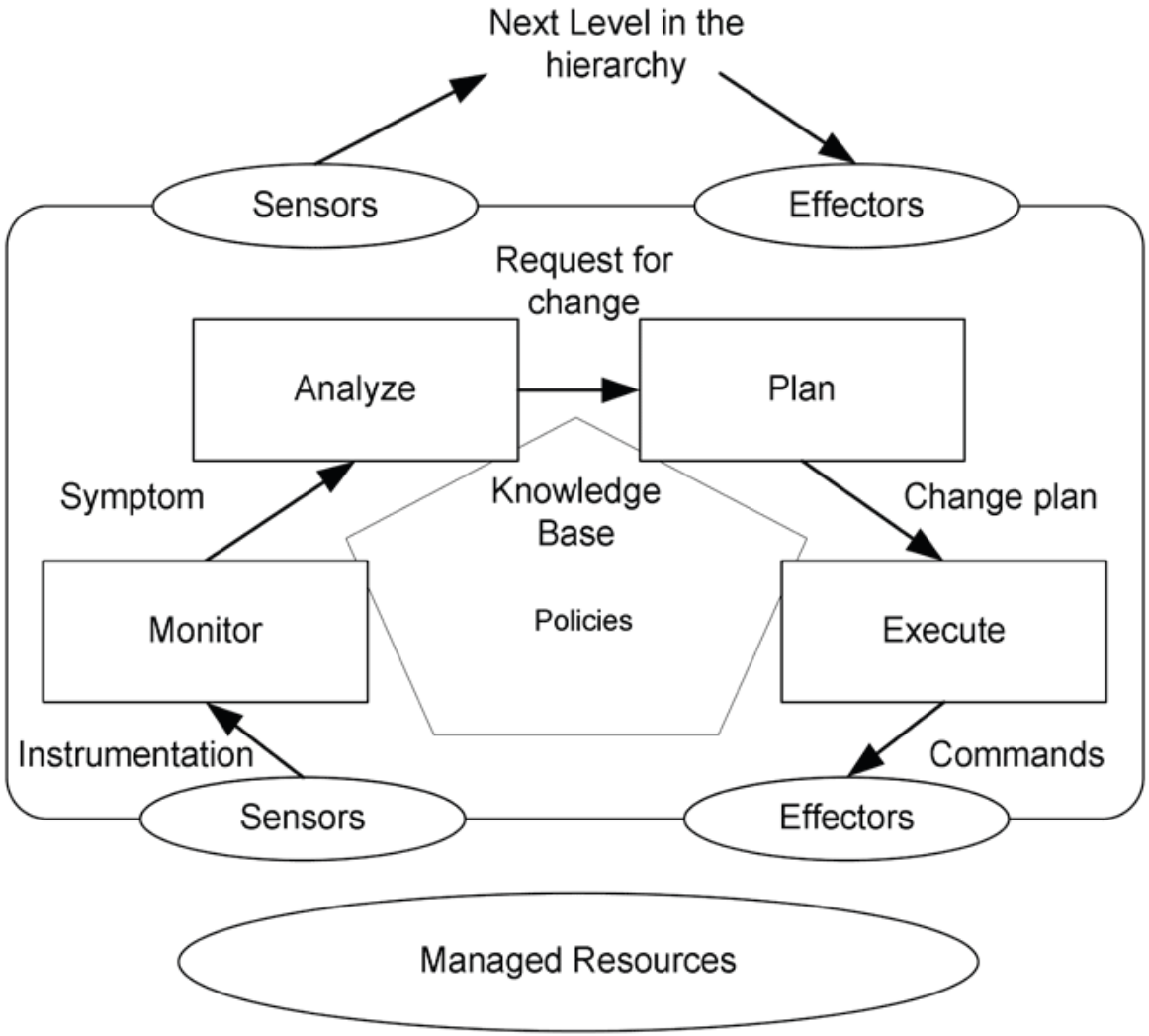
- *Just-in-time binding*. Bindings to particular computations, data, and hardware resources should be made only when actually required, preventing applications from creating bindings that would interfere with distribution or fault tolerance. Computations or data could be arbitrarily duplicated and bindings made to one instance would be equivalent to bindings to other instances.

- *Introspection*. The system should possess some aspects of self-examination and reflection. It should pervasively monitor itself and its applications, and reason about configuration and performance issues. Its models of its own configuration and operation should suggest opportunities for self-tuning as well as generate suggestions for physical configuration changes or upgrades that would improve performance.

The self-managing architecture

- The function of any autonomic capability is a control loop that collects details from the system and acts accordingly.
- The loop consists of four parts:
  - The monitor function: collects, aggregate, filter and reports info from the managed resource - symptom.
  - The analyze function: correlates and models complex situations; predicts future situations – request for change.
  - The plan function: uses policy information to command actions – change plan.
  - The execute function: controls the execution with dynamic updates.

Single autonomic manager



Where ?

- Advances in science and technology, e.g. in mobile and clouds, will eventually lead to a rich in resources and services, heterogeneous and highly dynamic Internet.
- One question that will need an answer is the balance between what runs on the device and what is provided by the Internet.
- OS research and development will play a significant role, together with advances in middleware systems.