A. Arpaci-Dusseau **Department of Computer Science**
CS739: Distributed Systems **University of Wisconsin, Madison**

A Comparison of Two Distributed Systems: Amoeba and Sprite

1 Introduction

- What was the motivation for this paper?

2 Design Philosophies

- What goals did two systems have in common?
- What workloads did Amoeba target? How did this impact where the developers focused their efforts? What did Sprite target and how did this impact development?
- The two systems both focused on sharing processors, but each system took a much different approach. For Amoeba where are the shared processors? What processes ran where? Why did the developers feel this model was best? What about for Sprite?
- The two systems both focused on sharing secondary storage and processors, but how they share storage is much different than how they share processors. Where is the secondary storage in each system? Why do you think that neither system treated secondary storage in the same way that they did processors?
- If you were developing a distributed OS today and choosing a processor allocation model, how would current technology trends influence you? What would the components look like?

3 Design Consequences

3.1 Kernel Architecture

- Why did Sprite choose a monolithic kernel architecture?
- Why did Amoeba choose a microkernel architecture?

3.2 Communication Mechanisms

- Judging only from the numbers in Table 1, what was the philosophy of Amoeba vs. Sprite for communication?

3.3 File System

- Both systems provide a globally-shared file system. What were Sprite's workload assumptions? How did that influence design? What were Amoeba's and how did it influence design? Which system do you think had more realistic assumptions?
- Both considered it important to be completely location transparent (i.e., each achieves the same results when running two processes on the same machine as on two different machines). How did each achieve system this?
- How does Amoeba further simplify memory and file system subsystems?
- Which system is likely to deliver better file performance to applications? Is this shown in Table 2?

3.4 Process Management

- To create a new process, Amoeba did not use the typical Unix `fork/exec` pair of system calls. What does it do instead? Why? What is the impact on the numbers in Table 3?
- How does Amoeba determine where a new process will run? How does Sprite? Are these approaches likely to scale with more processors? If add a shared processor pool to Sprite, how would the pool be used?
- How does each system ensure transparency for process placement and migration?

4 Concluding Questions

- Sprite had the explicit goal of remaining compatible with Unix, whereas Amoeba explicitly did not. Do you think researchers should worry about compatibility with existing systems?
- System projects often pay attention to technology trends. Which system do you think did a better job of predicting future technology?
- What were the strengths and weaknesses of Sprite?
- What were the strengths and weaknesses of Amoeba?
- What were the strengths and weaknesses of the paper?