Computer Science 5204
Operating Systems
Fall, 2010

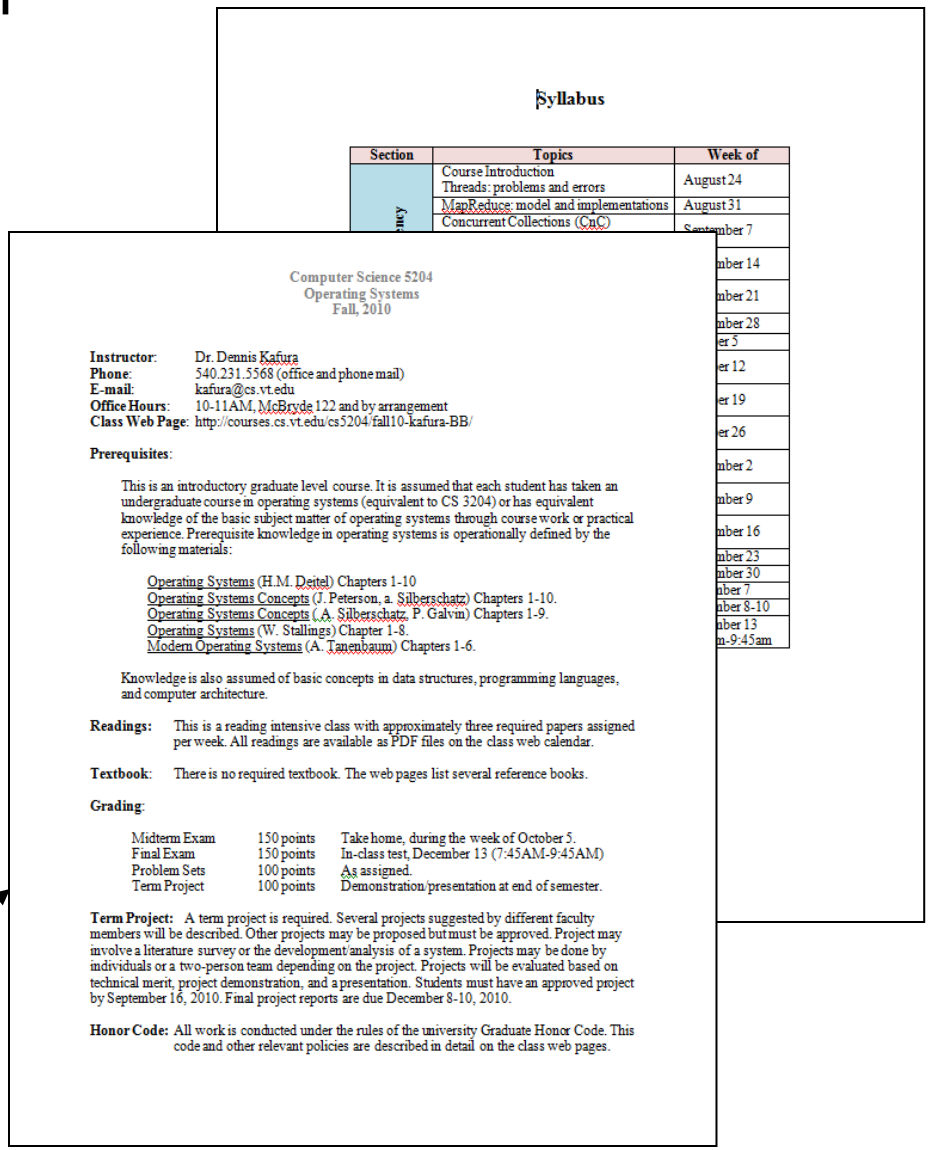
Dr. Dennis Kafura

Course Overview

Organization

- Material intensive
  - 35 +/- papers
  - 25 audio/video files
  - No required text
- Balance
  - Theory vs. technology
  - Contemporary vs. classic
  - Survey vs. depth
  - Centralized vs. distributed

Syllabus on web site



Syllabus

Section	Topics	Week of
	Course Introduction Threads: problems and errors	August 24
	MapReduce: model and implementations	August 31
	Concurrent Collections (CnC)	September 7
		mber 14
		mber 21
		mber 28
		er 5
		er 12
		er 19
		er 26
		mber 2
		mber 9
		mber 16
		mber 23
		mber 30
		mber 7
		mber 8-10
		mber 13 m-9:45am

Computer Science 5204
Operating Systems
Fall, 2010

Instructor: Dr. Dennis Kafura
Phone: 540.231.5568 (office and phone mail)
E-mail: kafura@cs.vt.edu
Office Hours: 10-11AM, McBryde 122 and by arrangement
Class Web Page: http://courses.cs.vt.edu/cs5204/fall10-kafura-BB/

Prerequisites:

This is an introductory graduate level course. It is assumed that each student has taken an undergraduate course in operating systems (equivalent to CS 3204) or has equivalent knowledge of the basic subject matter of operating systems through course work or practical experience. Prerequisite knowledge in operating systems is operationally defined by the following materials:

Operating Systems (H.M. Deitel) Chapters 1-10
Operating Systems Concepts (J. Peterson, a. Silberschatz) Chapters 1-10.
Operating Systems Concepts (A. Silberschatz, P. Galvin) Chapters 1-9.
Operating Systems (W. Stallings) Chapter 1-8.
Modern Operating Systems (A. Tanenbaum) Chapters 1-6.

Knowledge is also assumed of basic concepts in data structures, programming languages, and computer architecture.

Readings: This is a reading intensive class with approximately three required papers assigned per week. All readings are available as PDF files on the class web calendar.

Textbook: There is no required textbook. The web pages list several reference books.

Grading:

Midterm Exam	150 points	Take home, during the week of October 5.
Final Exam	150 points	In-class test, December 13 (7:45AM-9:45AM)
Problem Sets	100 points	As assigned.
Term Project	100 points	Demonstration/presentation at end of semester.

Term Project: A term project is required. Several projects suggested by different faculty members will be described. Other projects may be proposed but must be approved. Project may involve a literature survey or the development/analysis of a system. Projects may be done by individuals or a two-person team depending on the project. Projects will be evaluated based on technical merit, project demonstration, and a presentation. Students must have an approved project by September 16, 2010. Final project reports are due December 8-10, 2010.

Honor Code: All work is conducted under the rules of the university Graduate Honor Code. This code and other relevant policies are described in detail on the class web pages.

Course Web site

http://courses.cs.vt.edu/cs5204/fall10--kafura-BB

Major Topics and Themes

Topic	Theme					
	Atomicity	Order	Consistency	Theory	Protocols	Google
1. Concurrency	x	x		x		x
2. Security				x	x	
3. File systems	x	x	x			x
4. Fault tolerance	x	x	x		x	x

1. Concurrency

How can concurrent processing be structured on a single processor?

What are the problems and issues in developing systems with concurrent activities?

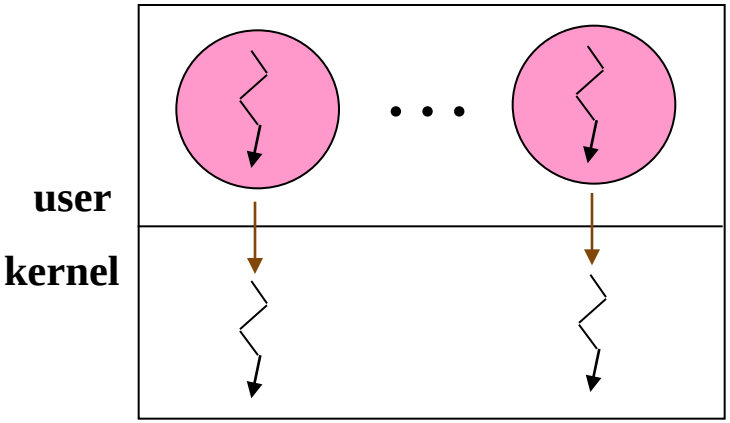
What are emerging models for concurrent programming?

How can the problems with multi-threaded programming be alleviated?

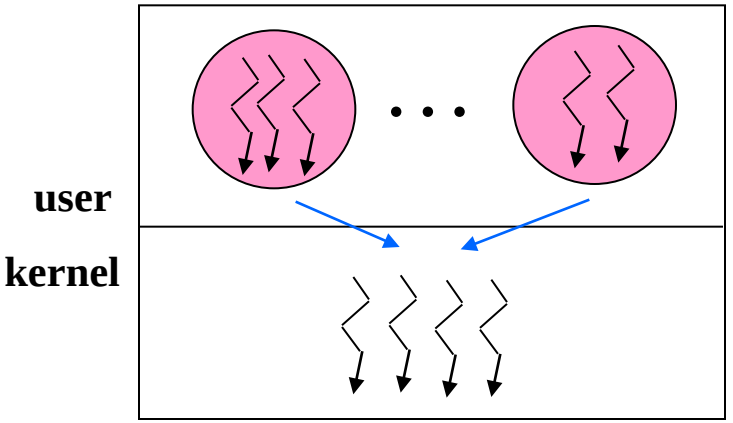
How can transaction-style semantics be supported locally in hardware or software?

How can concurrency and communication be represented formally?

Process vs. Thread Models



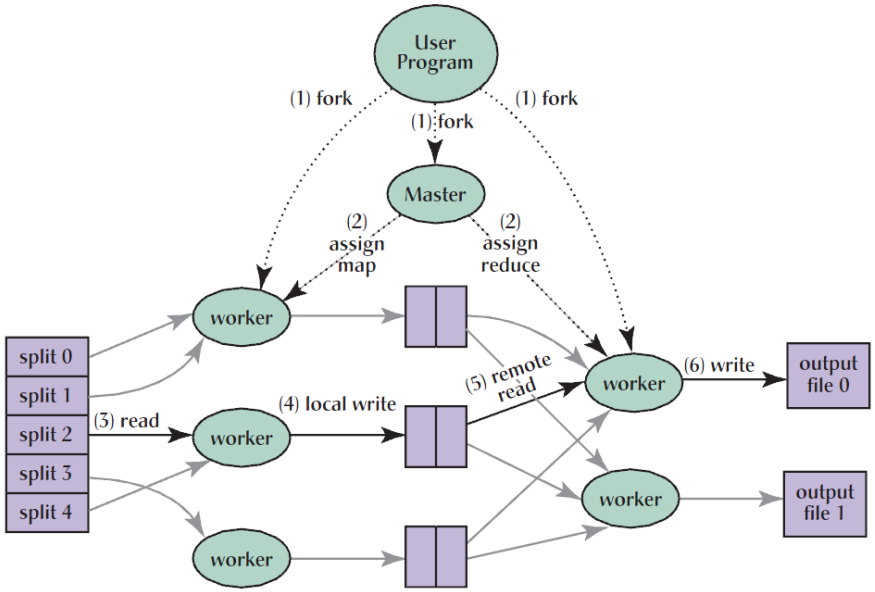
process-centered



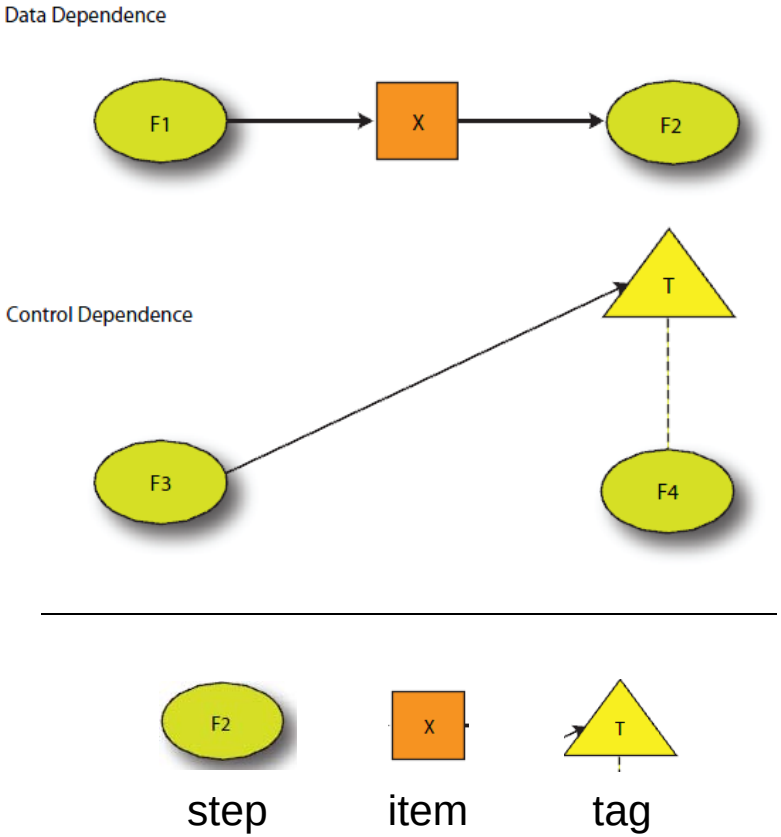
thread-centered

Models of Concurrent Computation

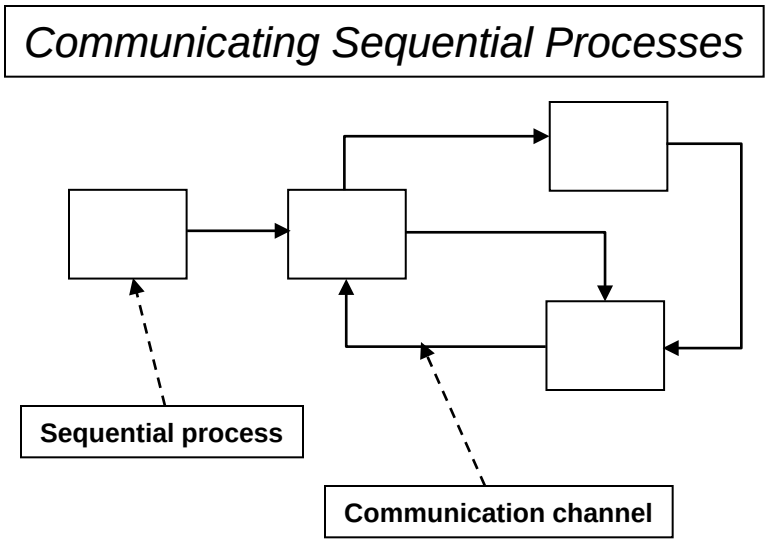
MapReduce

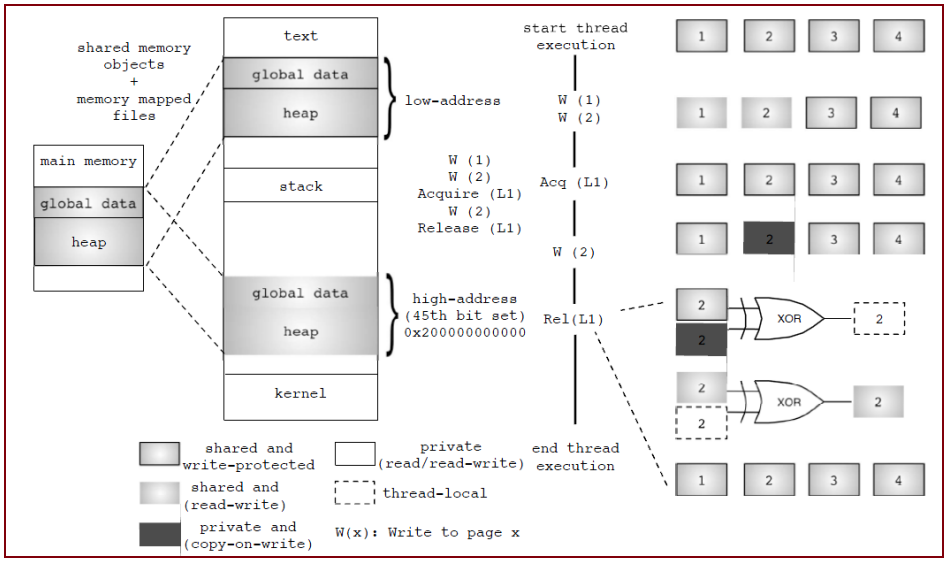


Concurrent Collections (CnC)

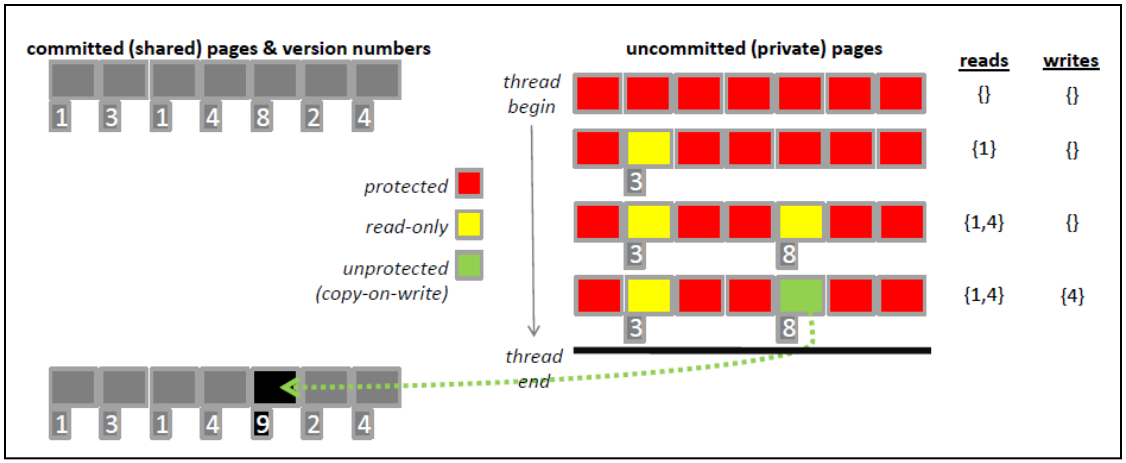


Thread-per-process models





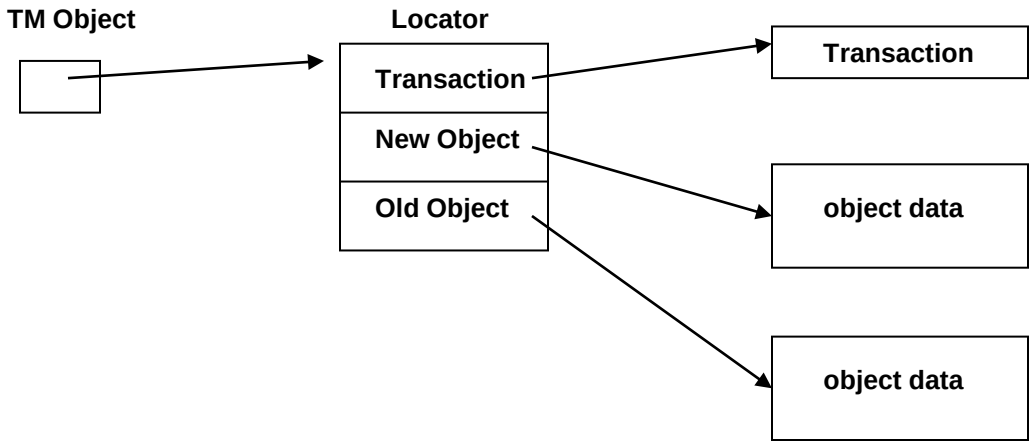
Sammati



Grace

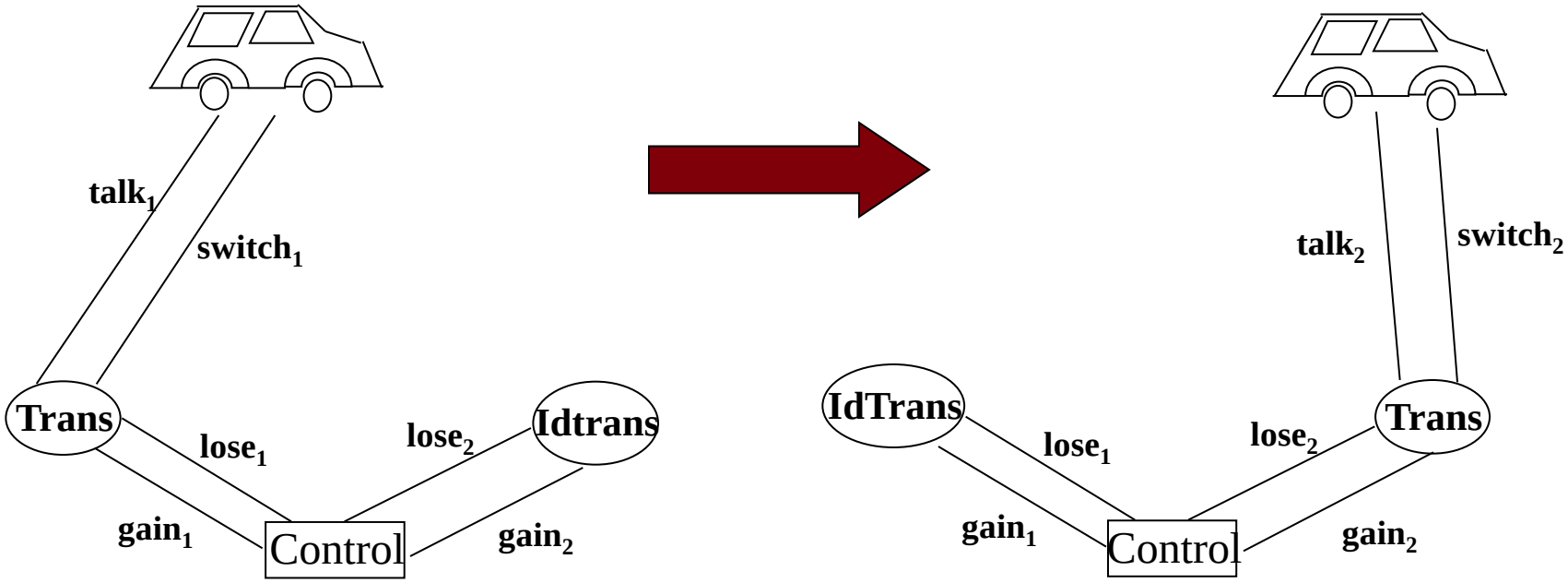
Supporting Transaction Semantics

```
repeat {

   BeginTransaction();     /* initialize transaction */
   <read input values>
   success = Validate();   /* test if inputs consistent */
   if (success) {
      <generate updates>
      success = Commit(); /* attempt permanent update */
      if (!success)
         Abort();/* terminate if unable to commit */
   }
   EndTransaction();       /* close transaction */

} until (success);
```



π-Calculus

An algebra that captures the notions of communication, interaction, and synchronization among concurrently executing entities.



2. Security

How can rights for access control be structured for effective use and management?

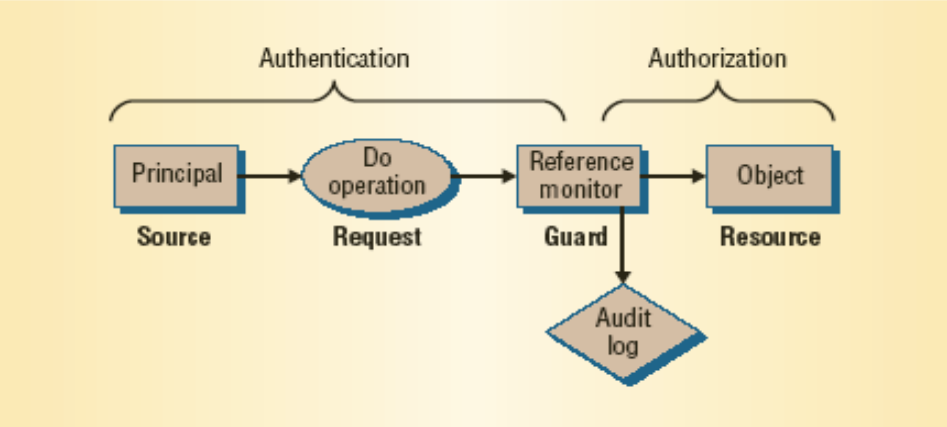
How can a digital document be "signed" so as to identify authorship?

How can communicating parties be confident of each other's identities?

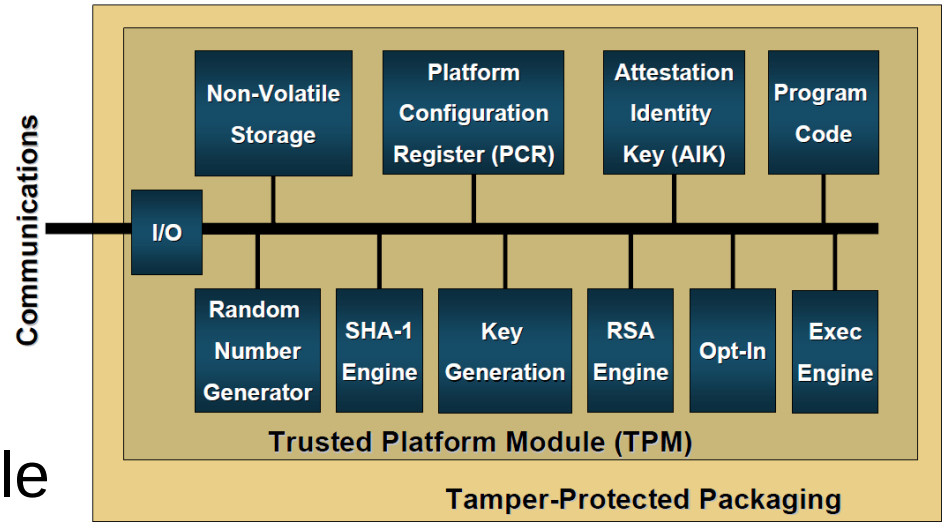
How can a platform attest to the validity of its execution environment?

How can access policies be expressed and enforced?

Security Overview

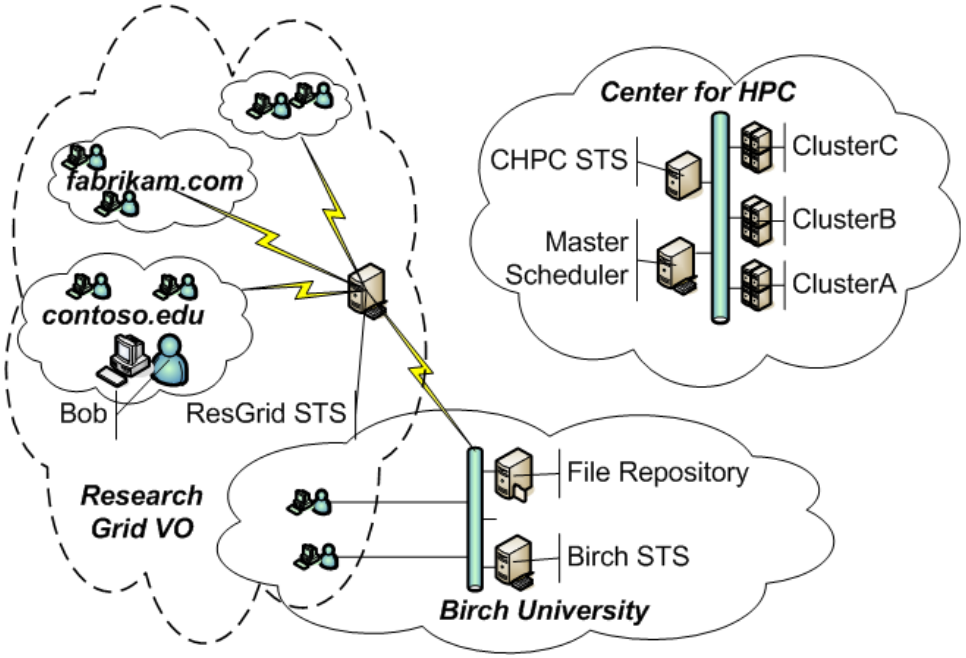


Models of Protection



Trusted Platform Module

Security in distributed systems



- Describe explicit trust relationships
- Express security token issuance policies
- Provide security tokens that contain identities, capabilities, and/or delegation policies
- Express resource authorization and delegation policies

3. File Systems

What does a "typical" file structure look like?

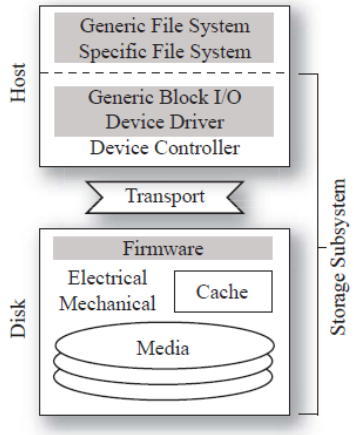
What problems can arise in the file sytems when systems fail?

How can file systems be structured to withstand (or more easily recover from) system failures?

How can file systems be structured to handle tera-bytes of information?

File Systems

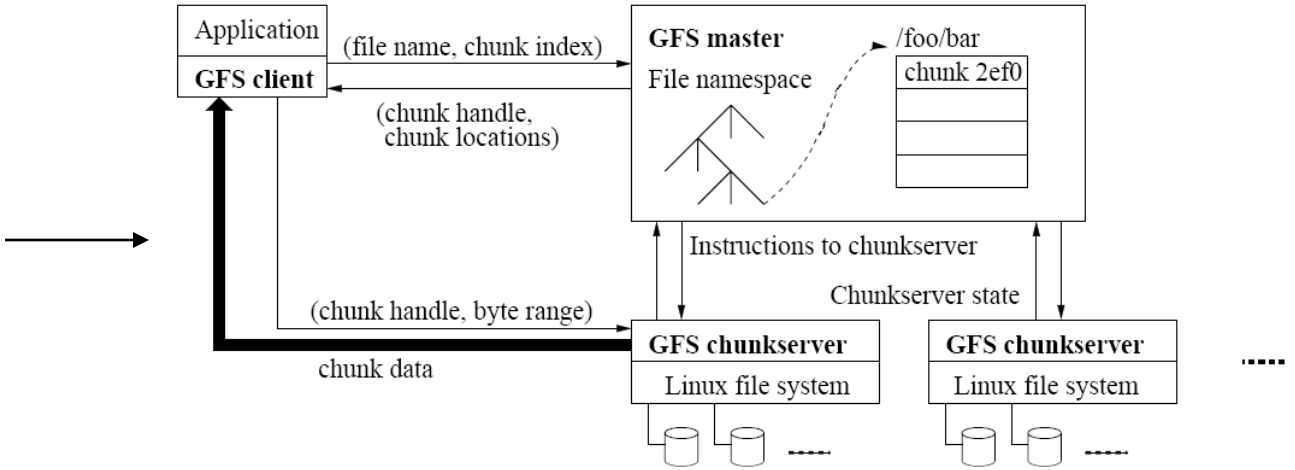
Storage Stack



Local File Systems Structures

- Log-structured file system
- Journaling and soft updates

Google File System



4. Fault Tolerance

How can events be ordered in a distributed system lacking a shared clock?

How can this ordering give rise to a form of virtual time?

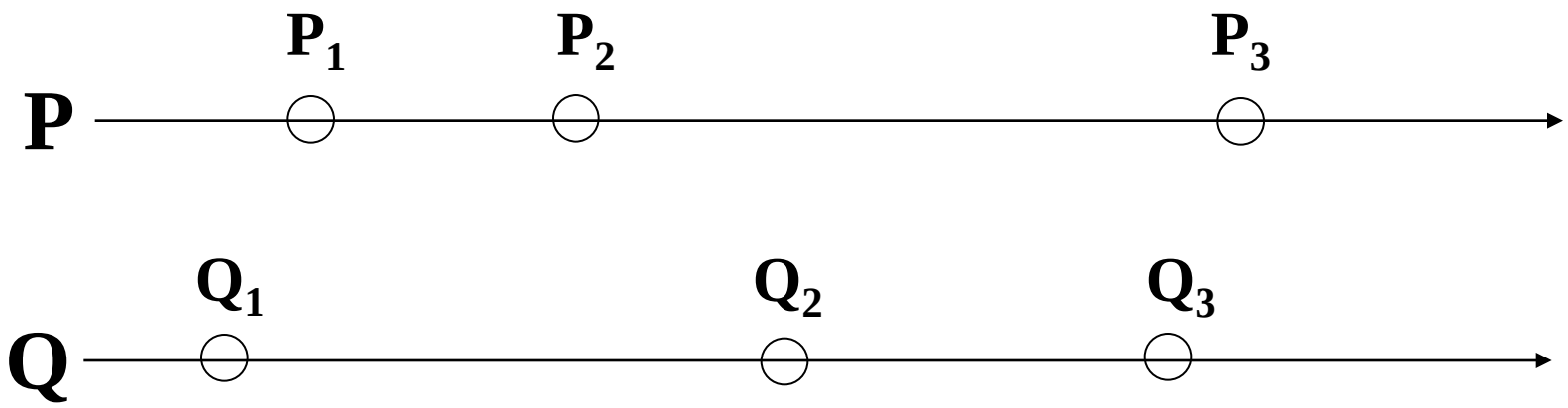
What are basic approaches to recovery from failure?

What is the taxonomy of strategies of "backward" recovery?

How can the state of system be captured so that it can be recovered in the event of failure?

How can distributed elements agree on commit to accepting a change in the system state?

Event Ordering

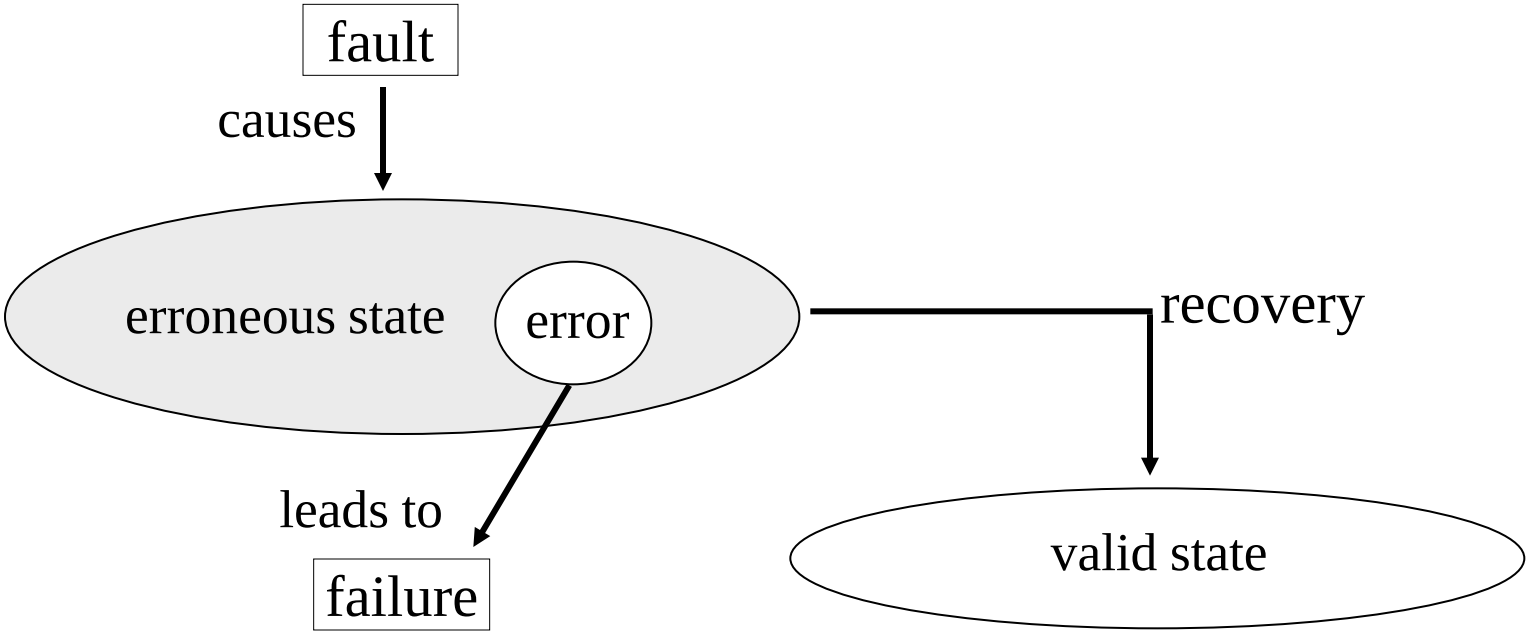


How can the events on P be related to the events on Q?

Which events of P "happened before" which events of Q?

When does it matter how we answer these questions?

Recovery



An error is a manifestation of a fault that can lead to a failure.

Failure Recovery:

- backward recovery
  - operation-based (do-undo-redo logs)
  - state-based (checkpoints)
- forward recovery