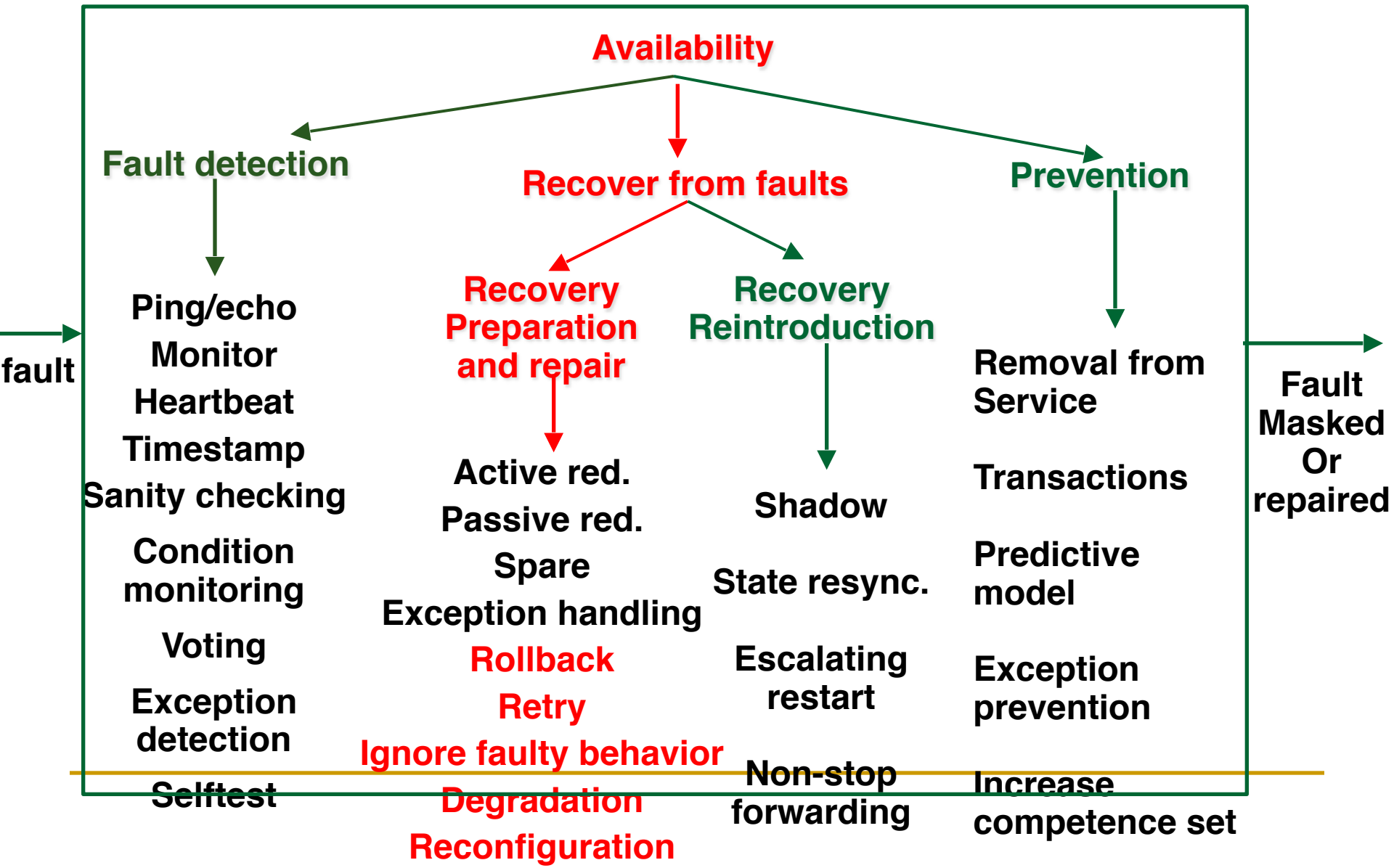
Tactics for Availability (2)

主讲教师：王灿
Email: wcan@zju.edu.cn
TA: 李奇平 liqiping1991@gmail.com
Course FTP: ftp://sa:sa@10.214.51.13

Availability Tactics Hierarchy



Rollback

- In case of failure, ***rollback*** reverts the system to a previous known good state, referred to as the "rollback line"
  - A copy of a previous good state can be saved by making a ***checkpoint*** of the system
  - Usually combines with active or passive redundancy
    - After a rollback, a standby version of the failed component is promoted to active status

Retry

- In networks and in server farms where failures are common and usually transient, the ***retry*** tactic will retry the failed operation which usually lead to success
  - A limit on the number of retries shall also be placed

Ignore Faulty Behavior

- Ignoring messages sent from a particular source when we determine that those messages are spurious
  - E.g. ignoring the message from a specific source in an DOS attack

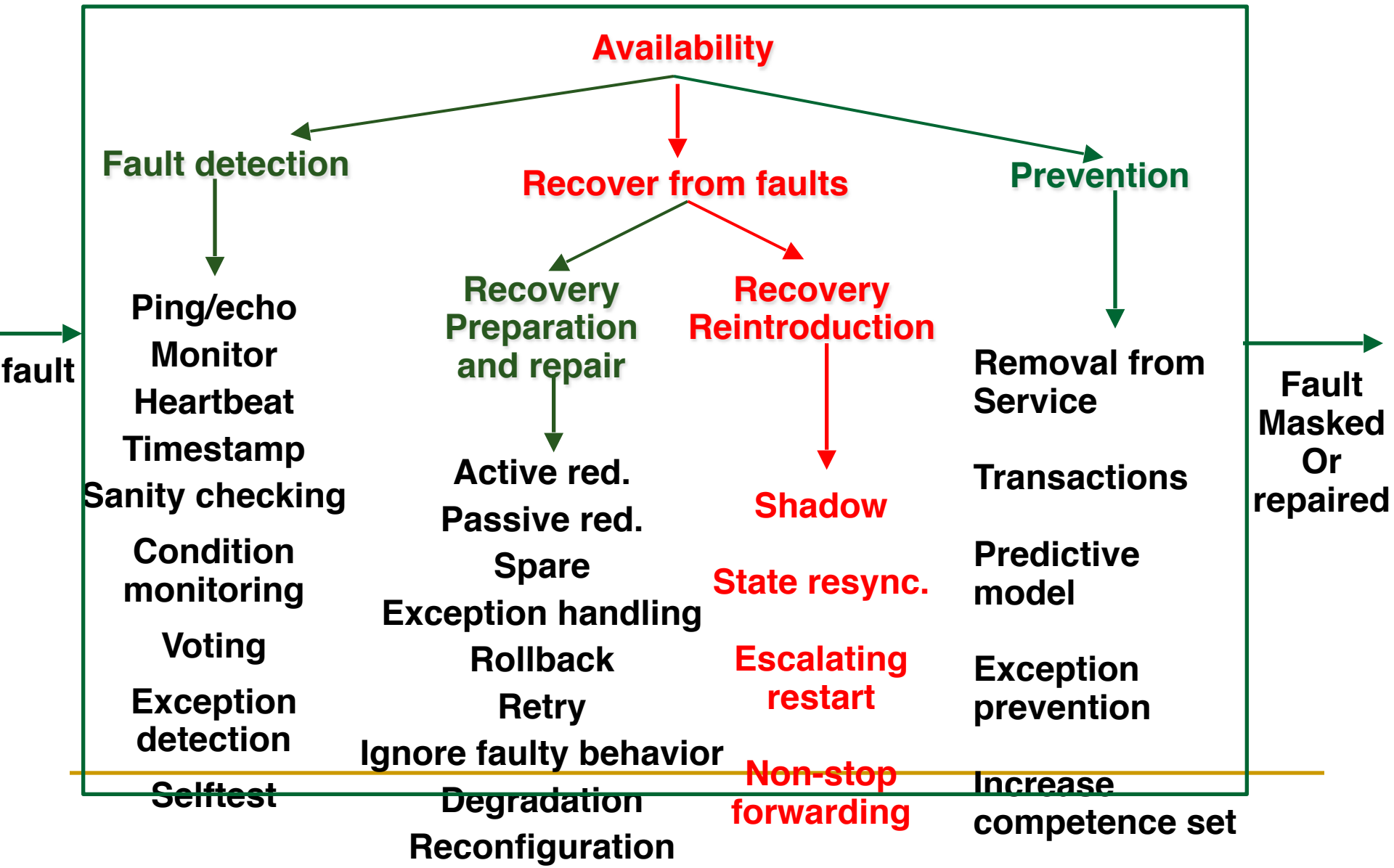
Degradation

- Maintaining the most critical system functions in the presence of component failures, dropping less critical functions

Reconfiguration

- Recovering from component failures by reassigning responsibilities to the (potentially restricted) resources left functioning, while maintaining as much functionality as possible

Availability Tactics Hierarchy

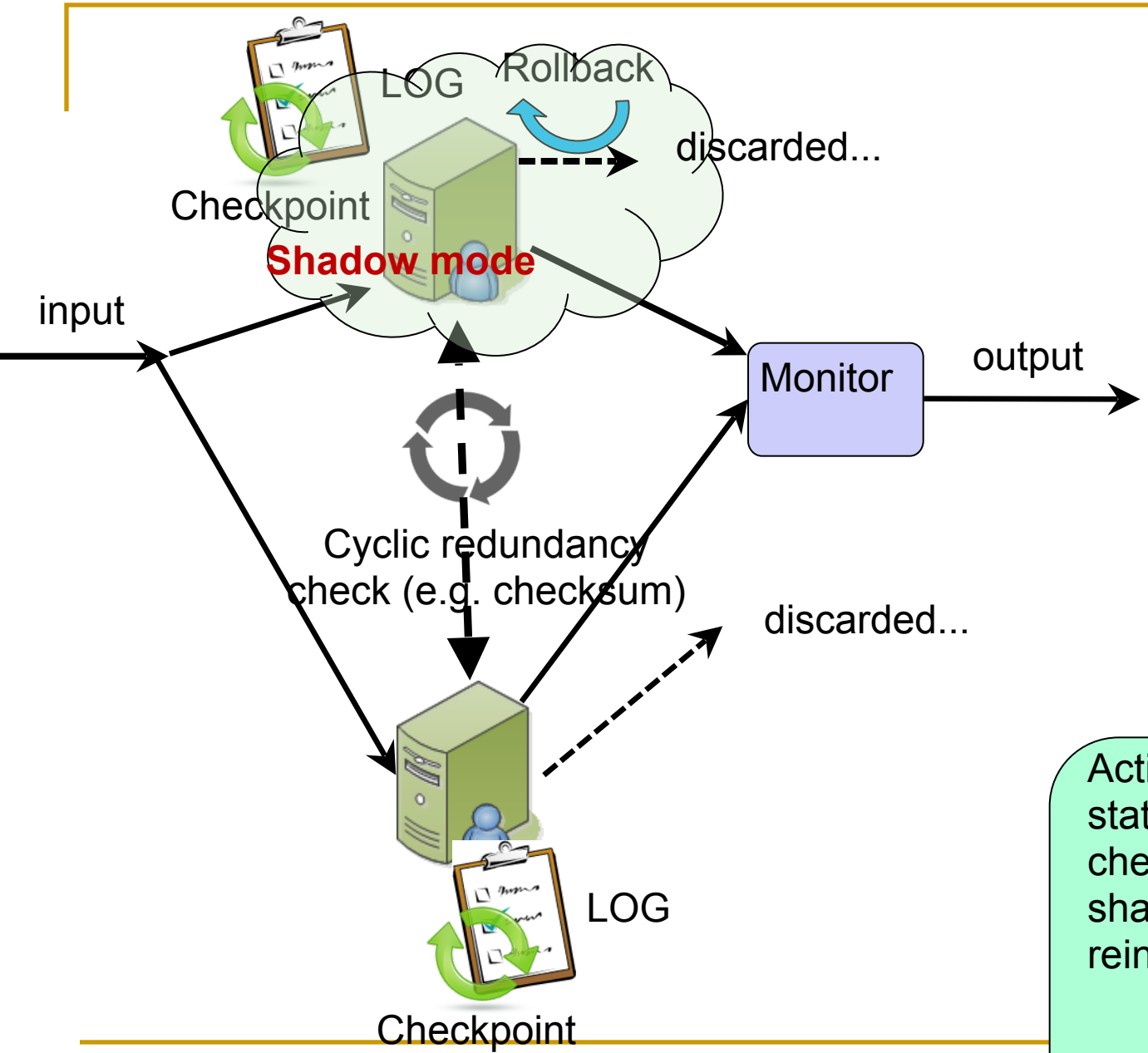


Shadow

- A previously failed or in-service upgraded component runs in a "shadow mode" temporarily prior to reverting the component back to an active role
  - Observe its behavior
  - Reinstatement can be done incrementally

State Resynchronization

- Usually used with the active redundancy or passive redundancy tactic
  - In active redundancy, state resynchronization is achieved by processing identical inputs in parallel
    - The states of the active and standby are periodically compared to ensure synchronization (e.g. checksum etc.)
  - In passive redundancy, state resynchronization is done by the active component periodically updating the state of the standby



Active redundancy with state resynchronization, checkpoint, rollback, shadow mode and reintroduction

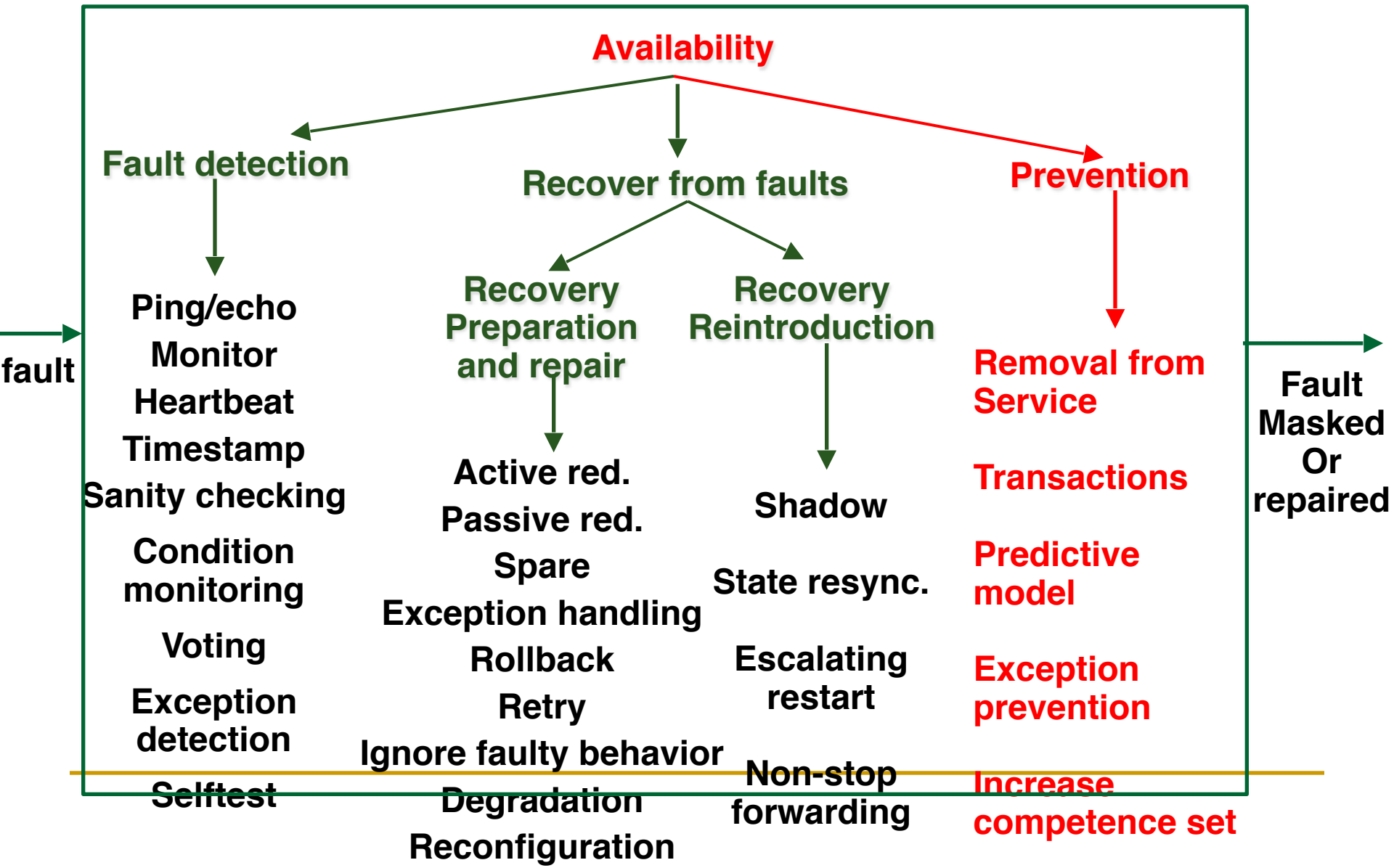
Escalating Restart

- A fault recovery tactic that minimizes the level of service affected by varying the granularity of the components restarted
  - E.g. an application running on a server provides multiple services and one of the service components failed. Available escalating restart options:
    - Restart the failed services component
    - Restart the whole affected service
    - Restart the application
    - Restart the server

Non-stop Forwarding (NSF)

- Used in router design. Router functions consist of
  - Supervisory (managing connectivity and routing information)
  - Packet routing
- When the supervisory part in a router fails, NSF tactic dictates that the packet routing part shall forward packets to the known neighboring routers
  - Meanwhile, the supervisory part is recovered using "graceful restart"

Availability Tactics Hierarchy



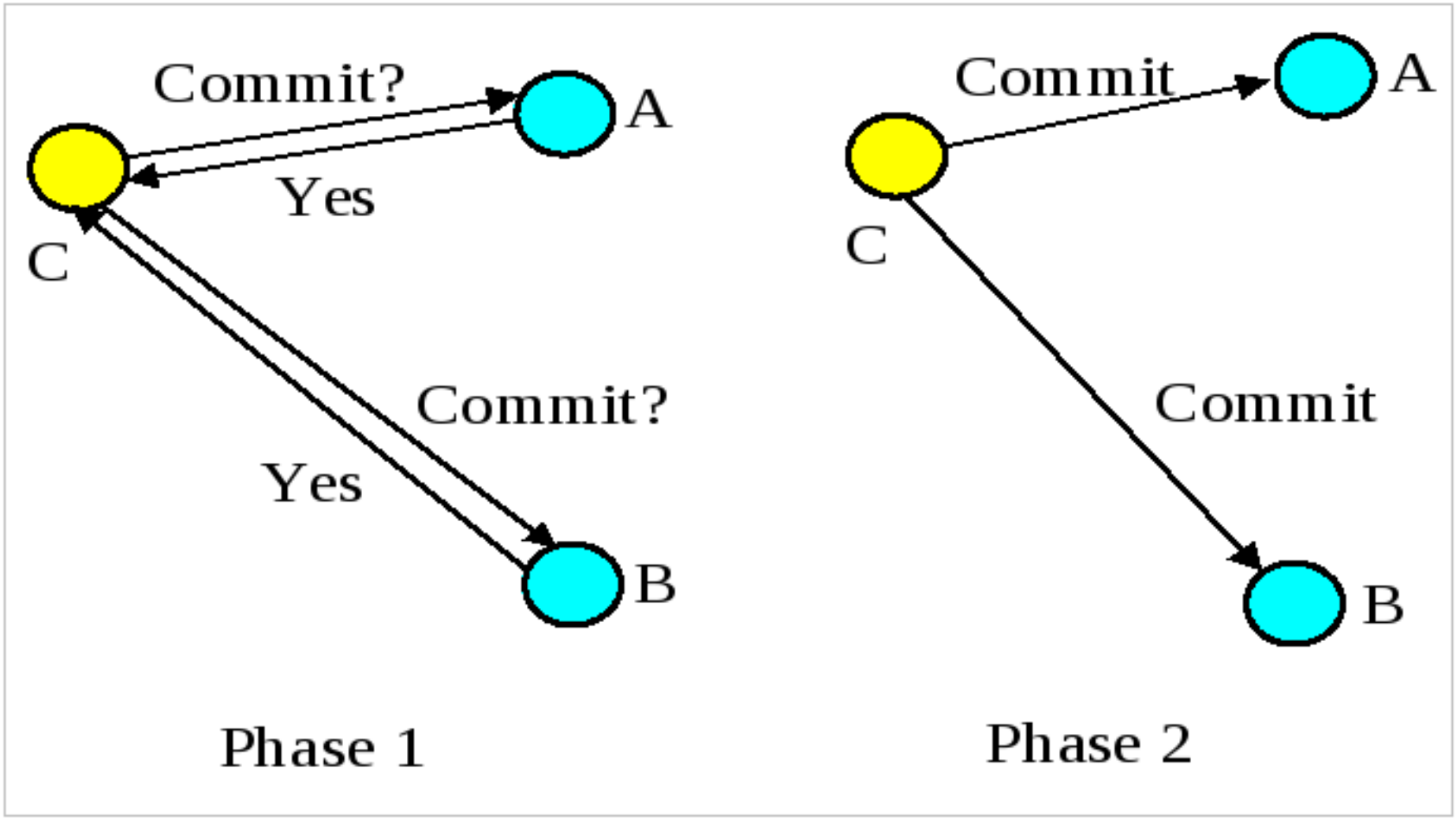
Removal from Service

- Temporarily taking a system component out of service to mitigate potential system failures (also known as *software rejuvenation*)
  - E.g. avoiding memory leaks, fragmentation, or soft errors in an unprotected cache

Transactions

- Using the transactional semantics to ensure the ACID properties of the asynchronous messages exchanged between distributed components
  - E.g. two-phase commit (2PC)

Two-phase Commit



Predictive Model

- A predictive model in fault detection is an monitor coupled with intelligent prediction
  - Operational performance metrics are used to predict the state of health of a system process
  - Corrective actions will be taken when potentially faulty conditions are detected

  - Examples
    - session establishment rate in an HTTP server
    - statistics for process state (in service, out of service, under maintenance, idle)
    - message queue length statistics

Exception Prevention

- Techniques that prevents system exceptions from occurring
- Examples
  - The use of exception classes
  - The use of wrappers to avoid dangling pointers and semaphore access violations
  - Smart pointers (prevent exceptions by bounds checking on pointers and avoid resource leak by automatic resource deallocation)

Increase Competence Set

- Employ more tolerance in component design, i.e. make it handle more cases and reduce the exceptions thrown

Architectural Design Support for Availability

- We check the architectural design and analysis process for availability from the following 7 aspects:
  1. Allocation of responsibilities
  2. Coordination model
  3. Data model
  4. Management of resources
  5. Mapping among architectural elements
  6. Binding time decisions
  7. Choice of technology

Allocation of Responsibilities

- For the system service with high availability requirements, ensure that your architectural design provides the following supports:
  - Detect an omission, crash, incorrect timing, or incorrect response
  - Log the fault
  - Notify appropriate entities (people or systems)
  - Disable the source of events causing the fault
  - Be temporarily unavailable
  - Fix or mask the fault/failure
  - Operate in a degraded mode

Coordination Model

- For the highly available parts in the system, ensure that coordination mechanisms:
  - Supporting detection of omission, crash, incorrect timing, or incorrect response (esp. guaranteed delivery, privileged transmission)
  - Logging; notification; disabling faulty sources; masking faults; degradation
  - Affected artifacts replacement (e.g. continued operation in server replacement)
  - Reliability under different conditions

Data Model

- Determine the possible faults and the data model for the highly available parts in the system
- Ensure the employed data abstractions, operations, and properties can be disabled , be temporarily unavailable, or be fixed or masked in the event of a fault
  - E.g. The write requests are cached if a server is temporarily unavailable and performed when the server is returned to service

Mapping among Architectural Elements

- Determine the faulty artifacts
- Ensure the mapping is flexible enough to support fault recovery; pay particular attention to two classes of mappings:
  - Runtime components to hardware infrastructure
    - E.g. processes → processors, when a processor fails
  - Redundancy of functionality, module → components

Resources Management

- Determine what critical resources are necessary to continue operating in the presence of a fault and ensure sufficient remaining resources for
  - Logging; notification; disabling faulty sources; masking faults; degradation..
- Determine the availability time for critical resources
  - E.g. ensure that input queues are large enough to buffer anticipated messages if a server fails

Binding Time Decisions

- Determine the architectural impact of the different bind choice.
- E.g. ensure the chosen availability strategy considers the following cases in late binding:
  - The fault detection tactic shall support all the bindings options
  - If the definition or tolerance of a fault is determined by late binding, ensure the recovery strategy is sufficient to handle all cases
  - The availability characteristics of the late binding mechanism itself

Choice of Technology

- Determine the available technologies and their characteristics
- Determine the availability characteristics of chosen technologies themselves
  - What faults can they recover from?
  - What faults might they introduce into the system?

Assignment

- Availability design
  - The assignment description has been uploaded to the FTP server.

Reading Assignment

- Read Chapter 6 of the textbook.