FAULT TOLERANT SYSTEM

CSE-508

Fault Tolerant System

Course Code:	CSE-508	Credits:	03
		CIE Marks:	90
Exam Hours:	03	SEE Marks:	60

Course Learning Outcome (CLOs): After Completing this course successfully, the student will be able to…

CLOs	Description
CLO1	Demonstrate an understanding of fault tolerance concepts, including faults, errors, and failures, and explain their impact on system reliability.
CLO2	Identify and apply various redundancy techniques (hardware, software, time, and information redundancy) to design fault-tolerant systems.
CLO3	Analyze dependability attributes such as reliability, availability, and safety, and use dependability evaluation techniques like Markov processes.
CLO4	Design and implement fault-tolerant systems using passive, active, and hybrid redundancy techniques for critical applications.
CLO5	Evaluate and model system reliability and availability using metrics such as MTTF, MTTR, MTBF, and fault coverage.

Summary of Course Content

Sl.	Course Content	HRs	CLOs
1	Introduction to Fault Tolerance, Faults, and Effects	2	CLO1
2	Dependability Attributes: Reliability, Availability, and Safety	3	CLO1, CLO3
3	Redundancy Techniques: Hardware, Software, Time, and Information Redundancy	4	CLO2
4	Reliability and Availability Evaluation Techniques	4	CLO3, CLO5
5	Markov Processes and State Transition Analysis	3	CLO3, CLO5
6	Passive Hardware Redundancy (TMR, NMR)	3	CLO2, CLO4
7	Active Hardware Redundancy (Standby Sparing, Duplication with Comparison)	3	CLO2, CLO4
8	Hybrid Redundancy: Static and Dynamic Techniques	3	CLO2, CLO4
9	Dependability Modeling and System Evaluation	4	CLO3, CLO5
10	Safety Analysis and Safety-Critical Applications	3	CLO3, CLO5
11	Fault-Tolerant Design Examples and Case Studies	4	CLO4, CLO5

- **Recommended Books:**

1. **Design of Fault-Tolerant Systems** by Elena Dubrova, Springer, 2013
2. **Fault-Tolerant Systems** by Israel Koren and C. Mani Krishna, Morgan Kaufmann, 2010
3. **Reliable Computer Systems: Design and Evaluation** by Daniel P. Siewiorek and Robert S. Swarz, CRC Press, 2011

Assessment Pattern

CIE- Continuous Internal Evaluation (90 Marks)

Bloom's Category Marks (out of 90)	Tests (45)	Assignments (15)	Quizzes (15)	Attendance (15)
Remember	5	03		
Understand	5	04	05	
Apply	15	05	05	
Analyze	10			
Evaluate	5	03	05	
Create	5			

SEE- Semester End Examination (60 Marks)

Bloom's Category	Test
Remember	7
Understand	7
Apply	20
Analyze	15
Evaluate	6
Create	5

Course Plan

Week No	Topics	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
1	Introduction to Fault Tolerance, Faults, and Effects	Lecture, Discussion	Quiz	CLO1
2	Dependability Attributes: Reliability, Availability, Safety	Lecture, Examples, Q&A	Assignment	CLO1, CLO3
3	Dependability Impairments: Faults, Errors, Failures	Lecture, Case Study	Quiz	CLO1
4	Redundancy Techniques: Hardware Redundancy	Lecture, Problem-Solving	Assignment	CLO2
5	Redundancy Techniques: Software, Time, and Information	Lecture, Demonstration	In-Class Exercise	CLO2
6	Reliability and Availability Evaluation Metrics	Lecture, Numerical Problem Solving	Quiz	CLO3, CLO5
7	Markov Processes and State Transition Analysis	Lecture, Worked Examples	Assignment	CLO3, CLO5
8	Passive Hardware Redundancy: TMR, NMR	Lecture, Case Studies	Quiz	CLO2, CLO4
9	Active Hardware Redundancy: Standby Sparing	Lecture, Group Discussion	Mid-Term Exam	CLO2, CLO4
10	Active Hardware Redundancy: Duplication with Comparison	Lecture, Demonstration	In-Class Exercise	CLO2, CLO4
11	Hybrid Redundancy Techniques	Lecture, Examples, Problem-Solving	Assignment	CLO2, CLO4
12	Dependability Modeling Techniques	Lecture, Group Work	Quiz	CLO3, CLO5
13	Safety Analysis and Safety-Critical Applications	Lecture, Case Studies	Assignment	CLO3, CLO5
14	Fault-Tolerant Design Examples and Case Studies	Lecture, Project Discussion	Group Presentation	CLO4, CLO5
15	Fault Detection and Coverage Techniques	Lecture, Problem Solving	In-Class Problem Solving	CLO2, CLO5
16	Reliability and Safety Evaluation Using Case Studies	Lecture, Hands-On Activity	Final Project Submission	CLO3, CLO5
17	Course Review and Final Exam	Recap, Q&A	Final Exam	CLO1, CLO2, CLO3, CLO4, CLO5

WEEK 1
SLIDES 6-14

Objectives

- understanding fault tolerance
  - faults and their effects (errors, failures)
  - redundancy techniques
  - evaluation of fault-tolerant systems
- balance
  - concepts, underlying principles
  - applications

Overview

- Introduction
  - definition of fault tolerance, applications
- Fundamentals of dependability
  - dependability attributes: reliability, availability, safety
  - dependability impairments: faults, errors, failures
  - dependability means
- Dependability evaluation techniques
  - common measures: failure rate, MTTF, MTTR
  - reliability block diagrams
  - Markov processes

Overview

- Redundancy techniques
  - space redundancy
    - hardware redundancy
    - information redundancy
    - software redundancy
  - time redundancy

INTRODUCTION TO FAULT TOLERANCE

Fault tolerance

fault-tolerance is the ability of a system to continue performing its function in spite of faults

broken connection	**hardware**
bug in program	**software**

Easily testable system

- Easily testable system is one whose ability to work correctly can be verified in a simple manner

Why do we need fault-tolerance?

- It is practically impossible to build a perfect system
  - suppose a component has the reliability 99.99%
  - a system consisting of 100 non-redundant components will have the reliability 99.01%
  - a system consisting of 10.000 components will have the reliability 36.79%
- It is hard to forsee all the factors

Redundancy

- Redundancy is the provision of functional capabilities that would be unnecessary in a fault-free environment
  - replicated hardware component
  - parity check bit attached to digital data
  - a line of program verfiying the correcntess of the resut

WEEK 2
SLIDES 15-23

History

- early computer systems
  - basic components had very low reliability
  - fault-tolerant techniques were need to overcome it
    - redundant structures with voting
    - error-detection and error correction codes

History

- early computer systems
  - EDVAC (1949)
    - duplicate ALU and compare results of both
    - continue processing if agreed, else report error
  - Bell Relay Computer (1950)
    - 2 CPU's
    - one unit begin executing the next instruction if the other encounts an error
  - IBM650, UNIVAC (1955)
    - parity check on data transfers

History

- Advent of transistors
  - more reliable components
  - led to temporary decrease in the emphasis on fault-tolerant computing
  - designers thought it is enough to depend on the improved reliability of the transistor to guarantee correct computations

History

- last decades
  - more critical applications
    - space programs, military applications
    - control of nuclear power stations
    - banking transactions
  - VLSI made the implementation of many redundancy techniques practical and cost effective
  - Other than hardware component faults need to be tolerated:
    - transient faults (soft errors) caused by environmental factors
    - software faults

Applications

- safety-critical applications
  - critical to human safety
    - aircraft flight control
  - environmental disaster must be avoided
    - chemical plants, nuclear plants
  - requirements
    - 99.99999% probability to be operational at the end of a 3-hour period

Applications

- mission-critical applications
  - it is important to complete the mission
  - repair is impossible or prohibitively expensive
    - Pioneer 10 was launched 2 March 1970, passed Pluto 13 June 1983
- requirements
    - 95% probability to be operational at the end of mission (e.g. 10 years)
    - may be degraded or reconfigured before (operator interaction possible)

Applications

- bisness-critical applications
  - users want to have a high probability of receiving service when it is requested
  - transaction processing (banking, stock exchange or other time-shared systems)
    - ATM: < 10 hours/year unavailable
    - airline reservation: < 1 min/day unavailable

Applications

- maintenance postponement applications
  - avoid unscheduled maintenance
  - should continue to function until next planned repair (economical benefits)
  - examples:
    - remotely controlled systems
    - telephone switching systems (in remote areas)

WEEK 3
SLIDES 24-43

Goals of fault tolerance

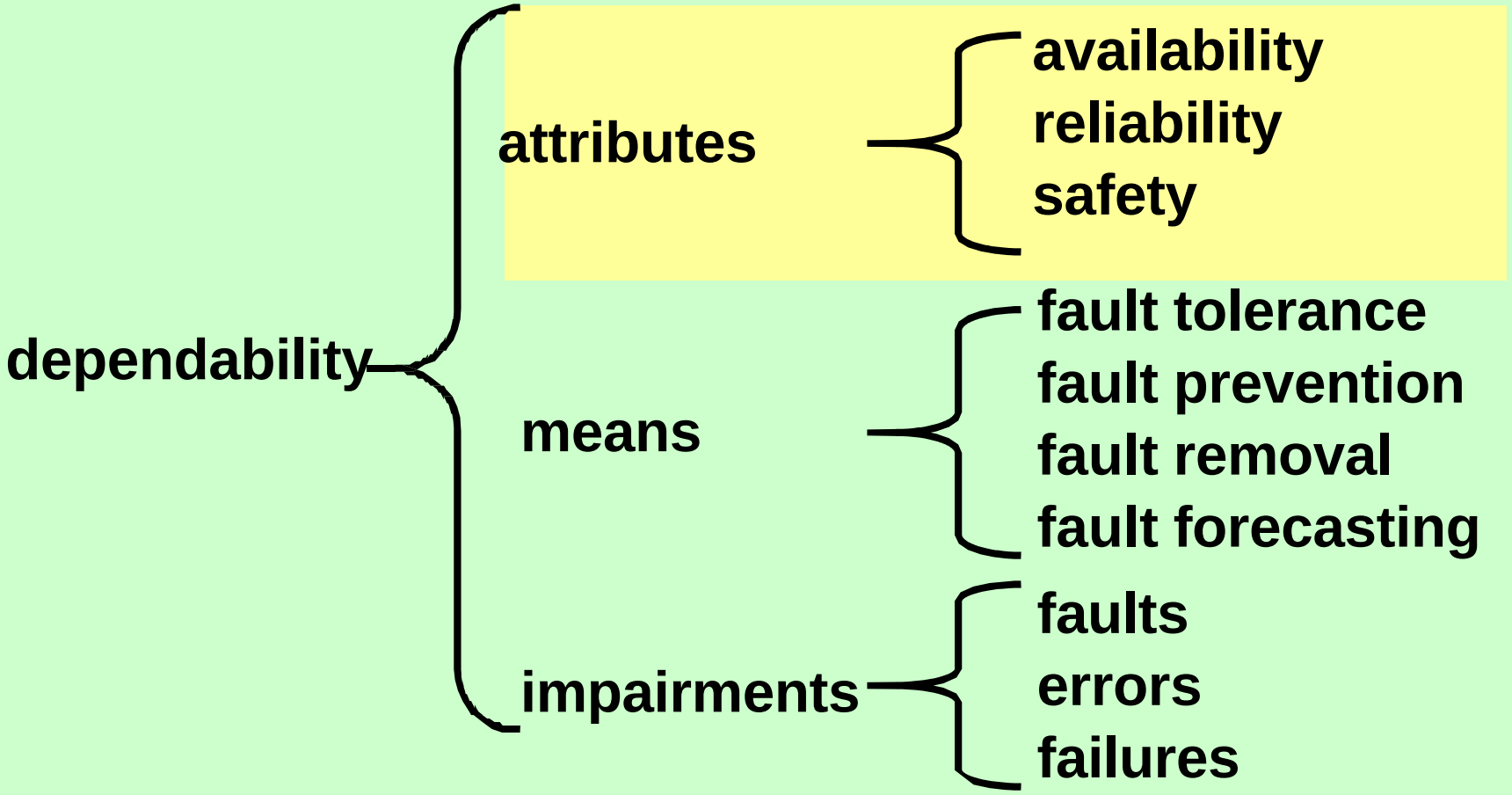
The main goal of fault tolerance is

to increase the dependability of a system

Dependability

Dependability
is the ability of a system to deliver its intended level of service to its users

Dependability tree



Reliability

- R(t) is the probability that a system operates without failure in the interval [0,t], given that it worked at time 0

- We need high reliability when:
  - even momentary periods of incorrect performance are unacceptable (aircraft, heart pace maker)
  - no repair possible (satellite, spacecraft)

High reliability examples

- airplane:
  - R(several hours) = 0.999 999 9 = 0.9_7
- spacecraft:
  - R(several years) = 0.95

Reliability versus fault tolerance

- Fault tolerance is a technique that can improve reliability, but
  - a fault tolerant system does not necessarily have a high reliability
  - a system can be designed to tolerate any single error, but the probability of such error to occur can be so high that the reliability is very low

Reliability versus fault tolerance

- A highly reliable system is not necessarily fault tolerant
  - a very simple system can be designed using very good components such that the probability of hardware failing is very low
  - but if the hardware fails, the system cannot continue its functions

How fault tolerance helps

- Fault tolerance can improve a system's reliability by keeping the system operational when hardware or software faults occur
  - a computer system with one redundant processor can be designed to continue working correctly even if one of the processors fails
  - **QUESTION:** Will a fault-tolerant system always be more reliable than an individual component?

Availability

- A(t) is the probability that a system is functioning correctly at the instant of time t

- depends on
  - how frequently the system becomes non-operational
  - how quickly it can be repaired

Steady-state availability

- Often the availability assumes a time-independent value after some initial time interval
- This value is called steady-state availability A_{ss}
- Steady-state availability is often specified in terms of downtime per year

  A_{ss} = 90%, downtime = 36.5 days/year

  A_{ss} = 99%, downtime = 3.65 days/year

Reliability versus availability

- reliability depends on an interval of time
- availability is taken at an instant of time
- a system can be highly available yet experience frequent periods of being non-operational as long as the length of each period is extremely short

High availability examples

- examples
  - transaction processing
    - ATM: $A_{ss}=0.9_3$ (< 10 hours/year unavailable)
    - banking: $A_{ss}=0.997$ (< 10 s/hour unavailable)
  - computing
    - supercomputer centres
      $A_{ss}=0.997$ (< 10 days/year unavailable)
  - embedded
    - telecom: $A_{ss}=0.9_5$ (< 5 min./year unavailable)

How fault tolerance helps

- Fault tolerance can improve a system's availability by keeping the system operational when a failure occur
  - a spare processor can perform the functions of the system, keeping its available for use, while the primary processor is being repaired

Safety

- Safety is the probability that a system will either perform its function correctly or will discontinue its operation in a safe way

- System is safe
  - if it functions correctly, or
  - if it fails, it remains in a safe state

High safety examples

- railway signalling
  - all semaphores red
- nuclear energy
  - stop reactor if a problem occur
- banking
  - don’t give the money if in doubt

Reliability versus safety

- Reliability is the probability that a system will perform its functions correctly
- Safety is the probability that a system will either work correctly or will stop in a manner that causes no harm

How fault tolerance helps

- Fault tolerance techniques can improve safety by turning a system off if a failure of a certain sort is detected
  - – in a nuclear power plant the reaction process should be stopped if some discrepancy is detected

Summary: attributes of dependability

- reliability:
  - continuity of service
- availability:
  - readiness for usage
- safety:
  - non-occurrence of catastrophic consequences

Next lecture

- Faults, error and failures
- Design philosophies to combat faults

Read chapters 1 and 2 of the text book

WEEK 4
SLIDES 44-53

EVALUATION TECHNIQUES

Two approaches

- Qualitative evaluation
  - aims to identify, classify and rank the failure modes, or event combinations that would lead to system failures
- Quantitative evaluation
  - aims to evaluate in terms of probabilities the attributes of dependability (reliability, availability, safety)

Common dependability measures

- failure rate
- mean time to failure
- mean time to repair
- mean time between failures
- fault coverage

Failure rate

- failure rate
  - expected # of failures per time-unit
  - example
    - 1000 controllers working at t_0
    - after 10 hours: 950 working
    - failure rate for each controller:
      0.005 failures / hour
      (50 failures / 1000 controllers) / 10 hours

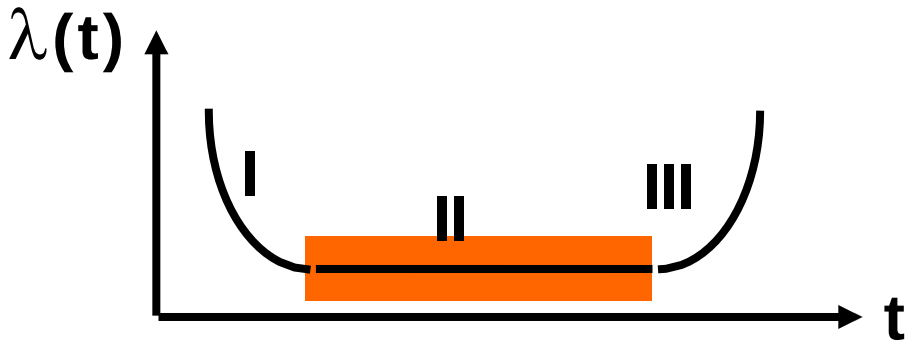
Failure rate and reliability

Reliability R(t) is the conditional probability that the system will perform correctly throughout [0,t], given that it worked at time 0

$$R(t) = \frac{N_{operating}(t)}{N_{operating}(t) + N_{failed}(t)}$$

Failure rate

- typical evolution of $\lambda(t)$ for hardware:



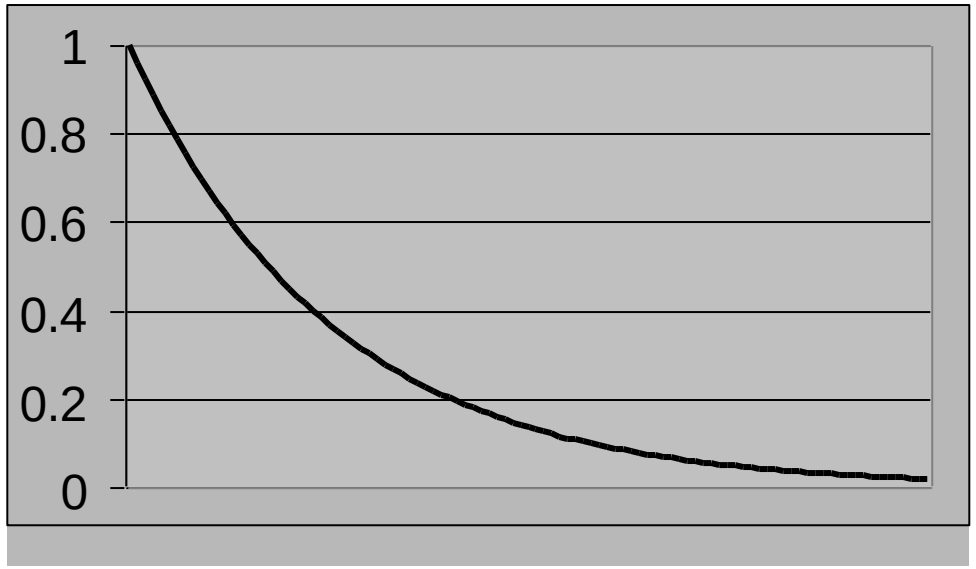
- bathtub: I infant mortality, II useful life, III wear-out
- for useful life period A = constant, the reliability is given by

$$R(t) = e^{-\lambda t}$$

Exponential failure law

$$R(t) = e^{-\lambda t}$$

If λ is constant, R(t) varies exponentially as a function of time



Time varying failure rate

- Failure rate is not always constant
  - software failure rate decreases as package matures
- Weibull distribution:

$$z(t) = \alpha\lambda(\lambda t)^{\alpha-1}$$

- if α=1, then z(t) = constant = λ
  if α>1, then z(t) increases as time increases
  if α<1, then z(t) decreases as time increases

$$R(t) = e^{-(\lambda t)^{\alpha}}$$

Failure rate calculation

- determined for components
  - systems: combination of components
  - λ of the system = sum of λ of the components
- determine A experimentally
  - slow
    - e.g. 1 failure per 100 000 hours (=11.4 years)
  - expensive
    - many components required for significance
- use standards for λ

WEEK 5
SLIDES 54-64

MTTF

- MTTF: mean time to failure
  - expected time until the first failure occurs
- If we have a system of N identical components and we measure the time t_i before each component fails, then MTTF is given by

$$MTTF = \frac{1}{N} . \sum_{i=1}^{N} t_i$$

MTTF

MTTF is defined in terms of reliability as:

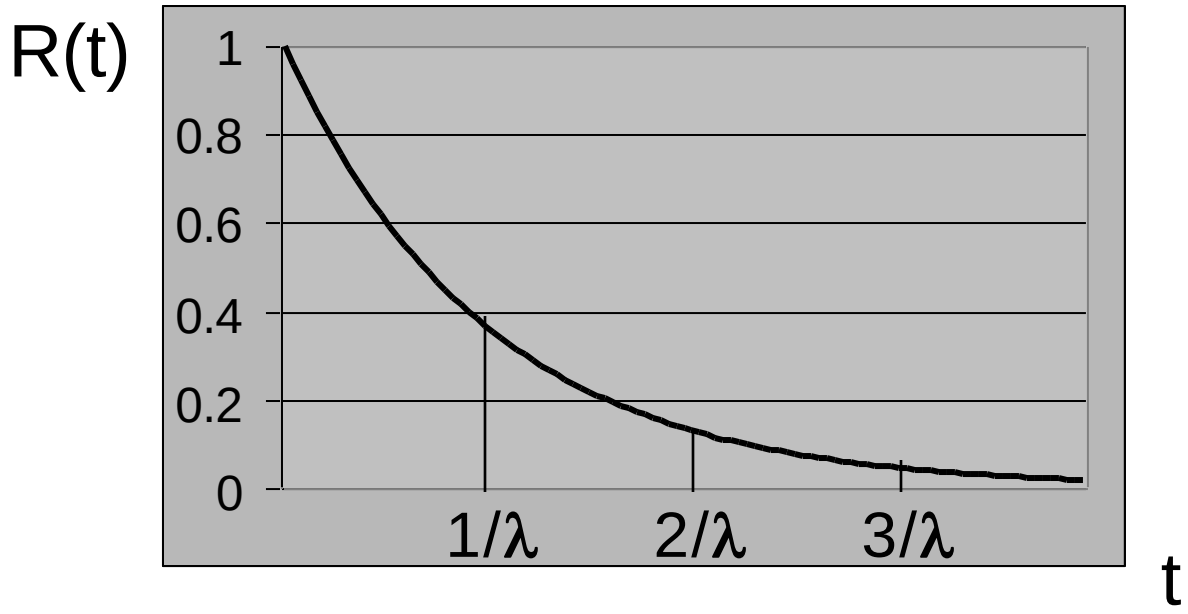
$$MTTF = \int_0^\infty R(t)dt$$

If R(t) obeys the exponential failure law, then MTTF is the inverse of the failure rate:

$$MTTF = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda}$$

MTTF

$$R(t) = e^{-\lambda t}$$



MTTF

- MTTF is meaningful only for systems which operate without repair until they experience a failure
- Most of mission-critical systems a undergo a complete check-up before the next mission
  - all failed redundant components are replaced
  - system is returned to fully operational state
- When evaluating reliability of such system, mission time rather then MTTF is used

MTTR

- MTTR: mean time to repair
  - expected time until repaired
- If we have a system of N identical components and i_{th} component requires time t_i to repair, then MTTR is given by

$$MTTR = \frac{1}{N} . \sum_{i=1}^{N} t_i$$

MTTR

- difficult to calculate
- determined experimentally
- normally specified in terms of repair rate repair rate μ, which is the average number of repairs that occur per time period

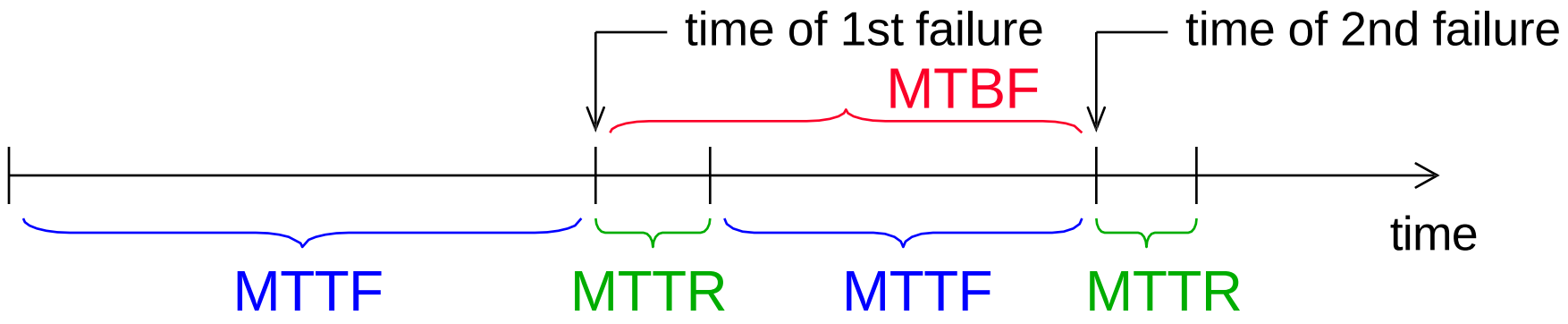
$$MTTR = \frac{1}{\mu}$$

MTTR

- Low MTTR requirement implies high operational cost
  - if hardware spares are kept on cite and the cite is maintained 24hr a day, MTTR=30min
  - if the cite is maintained 8hr 5 days a week, MTTR = 3 days
  - if system is remotely located MTTR = 2 weeks

MTBF

- MTBF: mean time between failures
  - functional + repair
  - MTBF = MTTF + MTTR
  - small time difference: MTBF ≈ MTTF
  - conceptual difference



Fault coverage

- Fault detection coverage is the conditional probability that, given the existence of a fault, the fault is detected
- Difficult to calculate
- Usually computed as

$$C = \frac{\text{number of faults which can be detected}}{\text{total number of faults}}$$

Example

- Suppose your circuit has 10 lines and you use single-stuck at fault as a model
- Then the total number of faults is 20
- Suppose you have 1 undetectable fault
- Then the coverage is

$$C = \frac{19}{20}$$

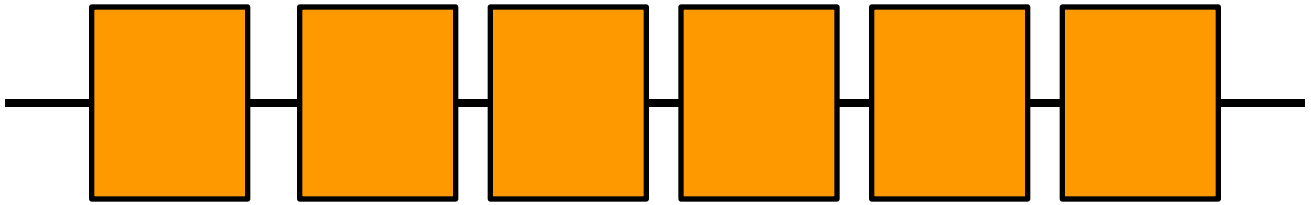
WEEK 6
SLIDES 65-71

Dependability modelling

- up to now: λ and R(t) for components
- systems are sets of components
- system evaluation approaches:
  - reliability block diagrams
  - Markov processes

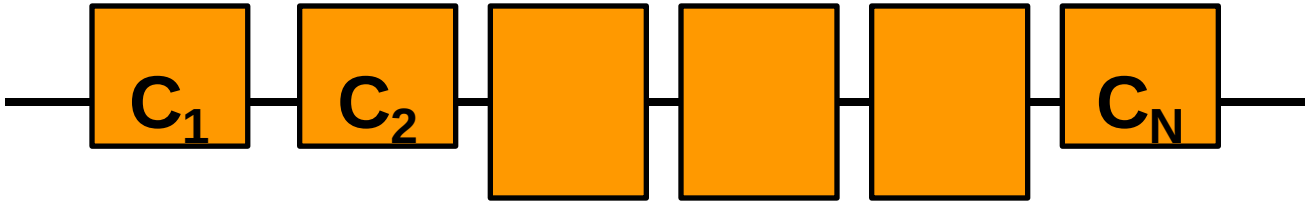
Serial system

- system functions
  if and only if all components function



reliability block diagram (RBD)

Serial system



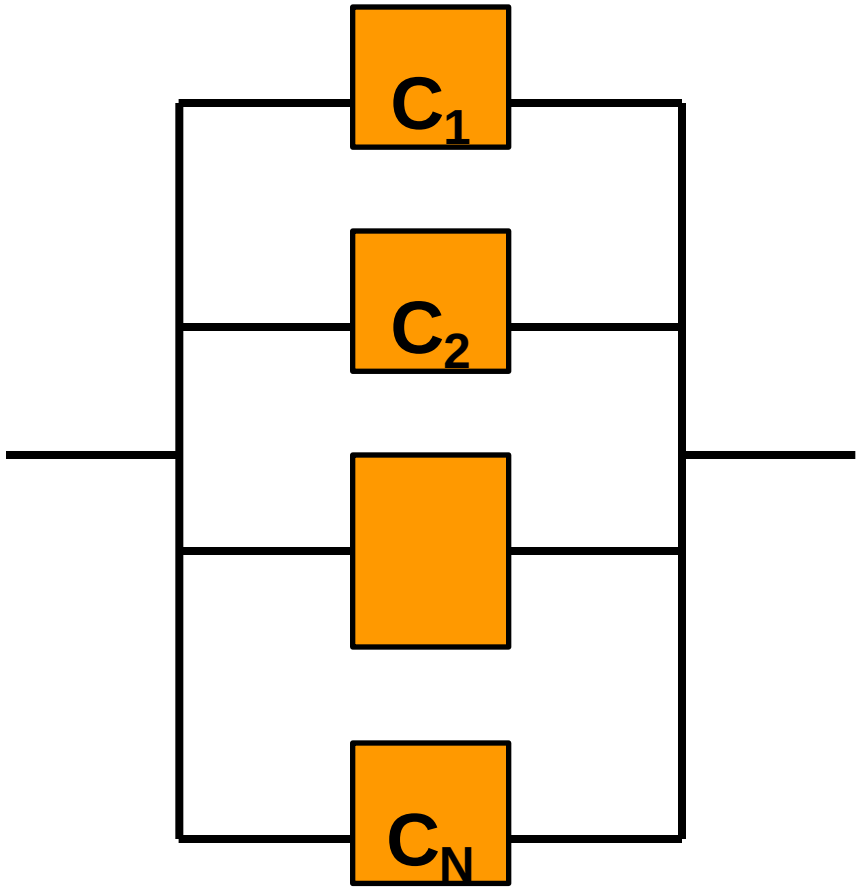
if C_i are independent:

$$R_{series}(t) = \prod R_i(t)$$

$$\lambda_{series} = \sum_{i=1}^{N} \lambda_i$$

Parallel system

- system works as long as one component works



Parallel system

unreliabity: Q(t) = 1 - R(t)

if C_i are independent: $$Q_{parallel}(t) = \prod_{i=1}^{N} Q_i(t)$$

$$R_{parallel}(t) = 1 - \prod_{i=1}^{N} (1 - R_i(t))$$

Reliability block diagram

- RBD
  - may be difficult to build
  - equations get complex
  - difficult to take coverage into account
  - difficult to represent repair
  - not possible to represent dependency between components

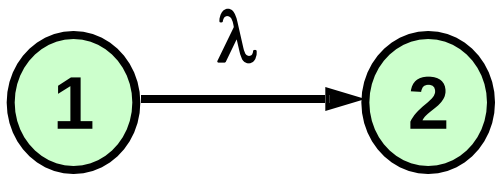
WEEK 7
SLIDES 72-78

Markov chains

- Markov chains
  - illustrated by state transition diagrams
- idea:
  - states
    - components working or not
  - state transitions
    - when components fail or get repaired

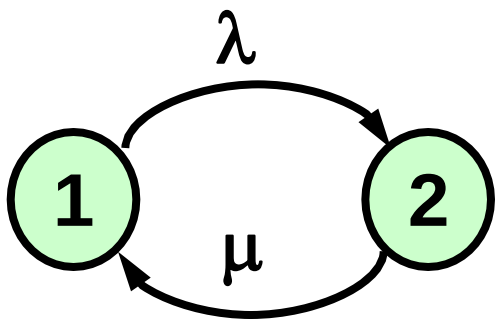
Single-component system, no repair

- Only two states
  - one operational (state 1) and one failed (state 2)
  - if no repair is allow, there is a single, non-reversible transition between the states (used in availability analysis)
  - label ″ corresponds to the failure rate of the component



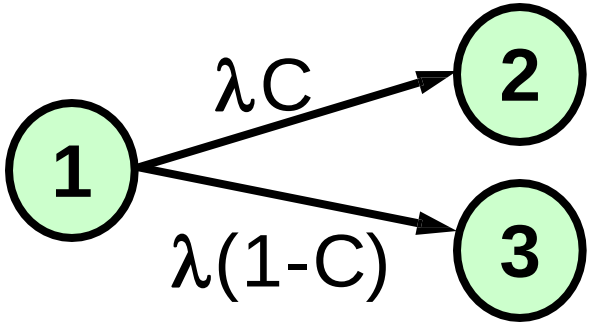
Single-component system with repair

- If repair is allowed (used in availability analysis)
  - then a transition between the failed and the operational state is possible
  - the label is the repair rate μ



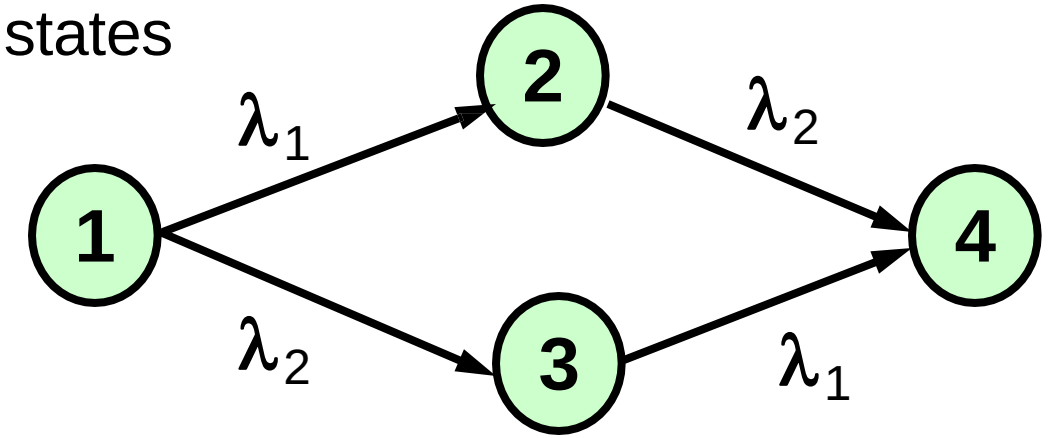
Failed-safe and failed-unsafe

- In safety analysis, we need to distinguish between failed-safe and failed-unsafe states
  - let 2 be a failed-safe state and 3 be a failed-unsafe state
  - the transition between the 1 and 2 depends on failure rate and the probability that, if a fault occurs, it is detected and handled appropriately (i.e. fault coverage C)
  - if C is the probability that a fault is detected, 1-C is the probability that a fault is not detected



Two-component system

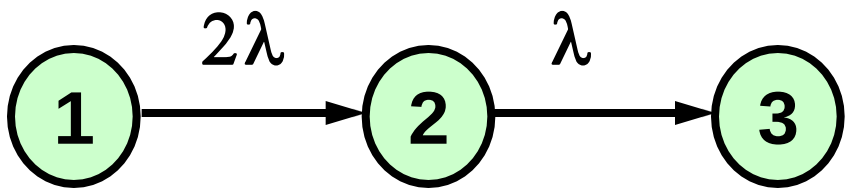
- Has four possible states
  - O O state 1
  - F O state 2
  - O F state 3
  - F F state 4



- Components are assumed to be independent and non-repairable
- If components are in serial
  - state 1 is operational state, states 2,3,4 are failed states
- If components are in parallel
  - states 1,2,3 are operational states, state 4 is failed state

State transition diagram simplification

- Suppose two components are in parallel
- Suppose $\mu_1 = \mu_2 = \mu$
- Then, it is not necessary to distinguish between between the states 2 and 3
  - both represent a condition where one component is operational and one is failed
  - since components are independent events, transition rate from state 1 to 2 is the sum of the two transition rates



WEEK 8
SLIDES 79-90

Markov chain analysis

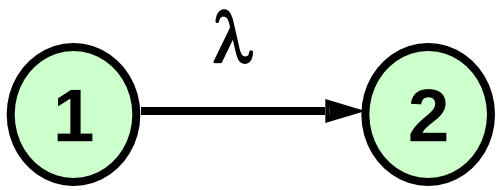
- The aim is to compute $P_i(t)$, the probability that the system is in the state i at time t
- Once $P_i(t)$ is known, the reliability, availability or safety of the system can be computed as a sum taken over all operating states
- To compute $P_i(t)$, we derive a set of differential equations, called state transition equations, one for each state of the system

Transition matrix

- State transition equations are usually presented in matrix form
- Transition matrix M has entries m_{ij}, representing the rates of transition between the states i and j
  - index i is used for the number of columns
  - index j is used for the number of rows

$$M = \begin{bmatrix} m_{11} & m_{21} \\ m_{12} & m_{22} \end{bmatrix}$$

Single-component system, no repair

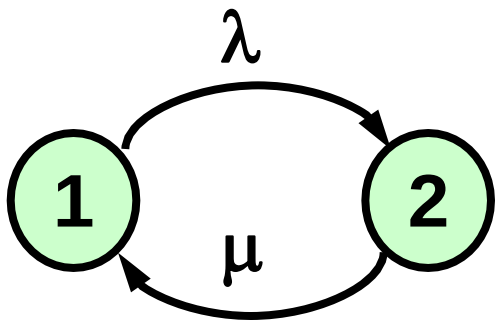


- Transition matrix M has the form:

$$M = \begin{bmatrix} -\lambda & 0 \\ \lambda & 0 \end{bmatrix}$$

- entries in each columns must sum up to 0
  - entries m_{ii}, corresponding to self-transitions, are computed as –(sum of other entries in this column)

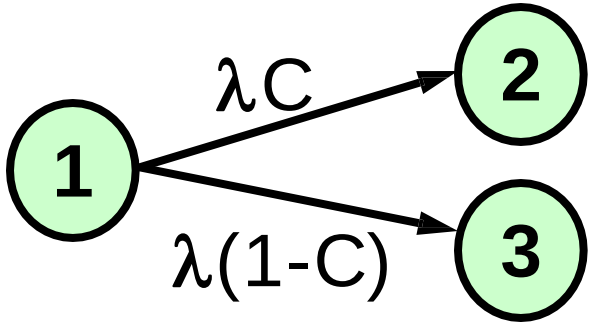
Single-component system with repair



- Transition matrix M has the form:

$$M = \begin{bmatrix} -\lambda & \mu \\ \lambda & -\mu \end{bmatrix}$$

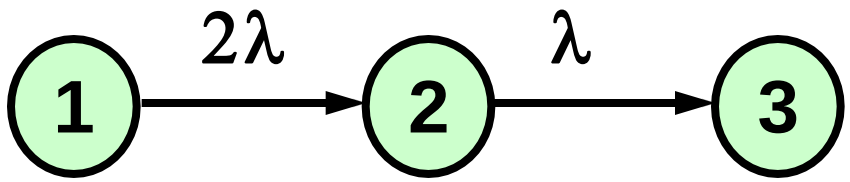
Single-component system, safety analysis



- Transition matrix M has the form:

$$M = \begin{bmatrix} -\lambda & 0 & 0 \\ \lambda C & 0 & 0 \\ \lambda(1-C) & 0 & 0 \end{bmatrix}$$

Two-component parallel system



- Transition matrix M has the form:

$$M = \begin{bmatrix} -2\lambda & 0 & 0 \\ 2\lambda & -\lambda & 0 \\ 0 & \lambda & 0 \end{bmatrix}$$

Important properties of matrix M

- Sum of the entries in each column is o
- Positive sign of an ij_{th} entry indicates that the transition originates from the i_{th} state
- In reliability analysis, M allows us to distinguish between the operational and failed states
  - – each failed state i has a zero diagonal element
    - m_{ii} (a failed state cannot leaved)

State transition equations

- Let P(t) be a vector whose i_{th} element is the probability $P_i(t)$, the probability that the system is in the state i at time t
- The matrix representation of a system of state transition equations is given by

$$\frac{d}{dt} P(t) = M \cdot P(t)$$

Two-component parallel system

- Using transition matrix derived earlier, we get:

$$\frac{d}{dt}\begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix} = \begin{bmatrix} -2\lambda & 0 & 0 \\ 2\lambda & -\lambda & 0 \\ 0 & \lambda & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

- This represents the following system of equations

$$\begin{cases} \frac{d}{dt}P_1(t) = -2\lambda P_1(t) \\ \frac{d}{dt}P_2(t) = 2\lambda P_1(t) - \lambda P_2(t) \\ \frac{d}{dt}P_3(t) = \lambda P_2(t) \end{cases}$$

Solving state transition equations

- By solving these equations, we get

$$P_1(t) = e^{-2\lambda t}$$

$$P_2(t) = 2e^{-\lambda t} - 2e^{-2\lambda t}$$

$$P_3(t) = 1 - 2e^{-\lambda t} + e^{-2\lambda t}$$

- Since the $P_i(t)$ are known, we can compute the reliability of the system as a sum of probabilities taken over all operating states

$$R_{parallel}(t) = P_1(t) + P_2(t) = 2e^{-\lambda t} - e^{-2\lambda t}$$

Comparison to RBD result

- Since $R = e^{-\lambda t}$, the previous equation can be written as

$$R_{parallel}(t) = 2R - R^2$$

- which agrees with the expression derived using RBD
- two results are the same because we assumed that the failure rates of the two components are independent

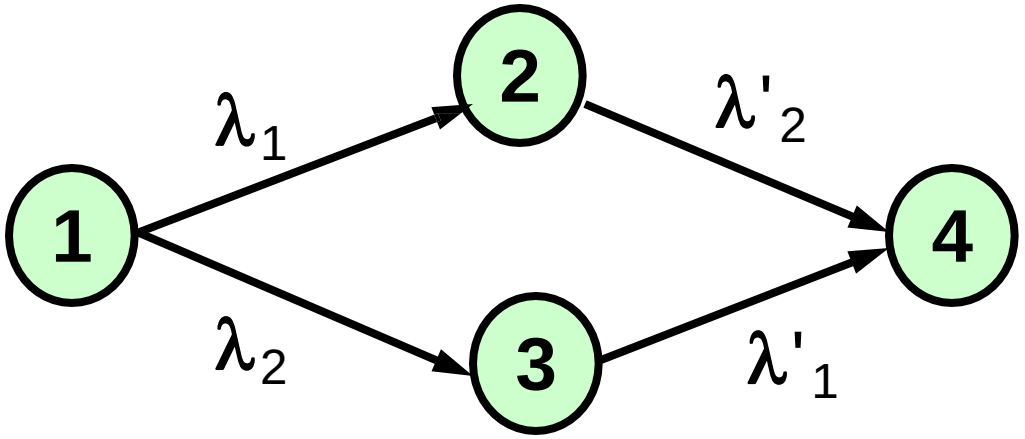
WEEK 9
SLIDES 91-102

Dependant component case

- The value of Markov chains become evident when component failures cannot be assumed to be independent
  - load-sharing components
  - examples: electrical load, mechanical load, information load
- If two components share the same load and one fails, the additional load on the second component increases its failure rate

Parallel system with load sharing

- As before, we have four states, but after the 1st component failure, the failure rate of the 2nd component increases



Parallel system with load sharing

- State transition equations are:

$$\frac{d}{dt}\begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \end{bmatrix} = \begin{bmatrix} -\lambda_1-\lambda_2 & 0 & 0 & 0 \\ \lambda_1 & -\lambda'_2 & 0 & 0 \\ \lambda_2 & 0 & -\lambda'_1 & 0 \\ 0 & \lambda'_2 & \lambda'_1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \end{bmatrix}$$

$$\begin{cases} \frac{d}{dt}P_1(t) = (-\lambda_1-\lambda_2)P_1(t) \\ \frac{d}{dt}P_2(t) = \lambda_1 P_1(t) - \lambda'_2 P_2(t) \\ \frac{d}{dt}P_3(t) = \lambda_2 P_1(t) - \lambda'_1 P_3(t) \\ \frac{d}{dt}P_4(t) = \lambda'_2 P_2(t) + \lambda'_1 P_3(t) \end{cases}$$

Effect of the load

- If $\lambda'_1 = \lambda_1$ and $\lambda'_2 = \lambda_2$, the equation of load sharing parallel system reduces to well- known

$$R_{parallel}(t) = 2e^{-\lambda t} - e^{-2\lambda t}$$

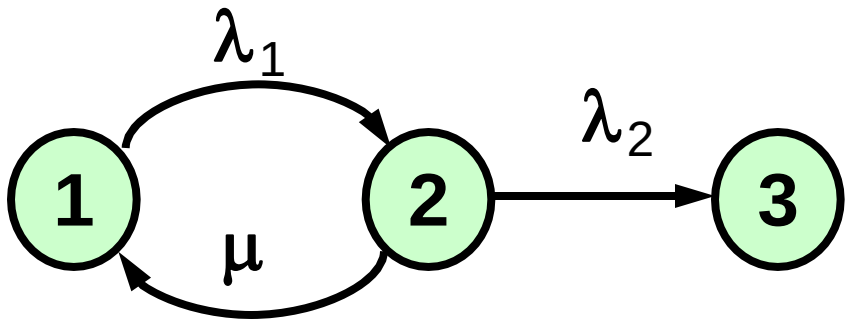
Availability evaluation

- Difference with reliability analysis:
  - in reliability analysis components are allowed to be repaired as long as the system has not failed
  - in availability analysis components can also be repaired after the system failure

Two-component standby system

- First component is primary
- Second is held in reserve and only brought to operation if the first component fails
- We assume that
  - fault detection unit which detect failure of the primary component are replace is with standby is perfect
  - standby component cannot fail while in the standby mode

State transition diagram for reliability analysis with repair



$$M = \begin{bmatrix} -\lambda_1 & \mu & 0 \\ \lambda_1 & -\lambda_2-\mu & 0 \\ 0 & \lambda_2 & 0 \end{bmatrix}$$

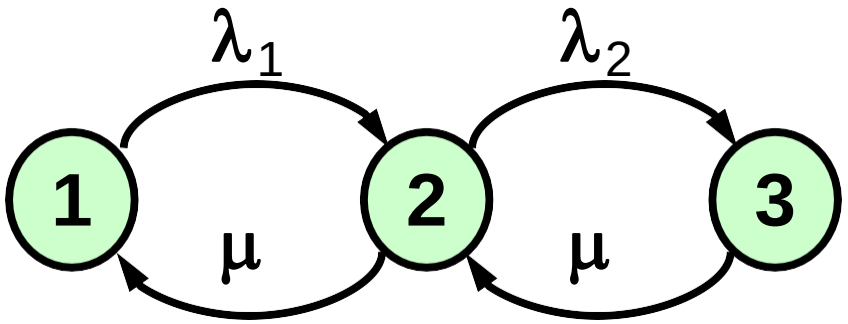
state 1: both OK

state 2: primary failed and replaced by spare

state 3: both failed

Repair replaces a broken component by a working one.

State transition diagram for availability analysis with repair

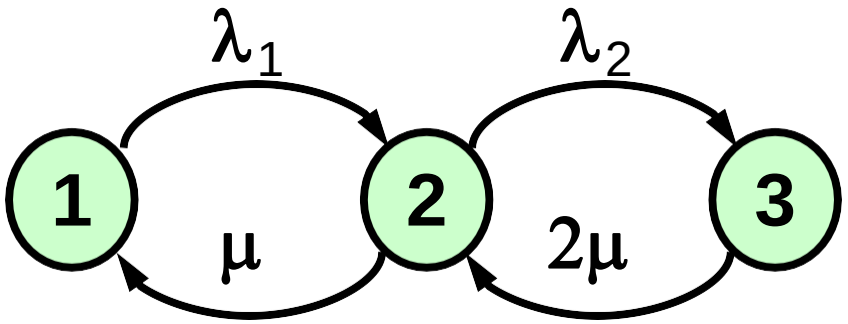


$$M = \begin{bmatrix} -\lambda_1 & \mu & 0 \\ \lambda_1 & -\lambda_2-\mu & \mu \\ 0 & \lambda_2 & -\mu \end{bmatrix}$$

States are the same.

Repair replaces a broken component by a working one. Here we assume that there is only one repair team.

State transition diagram for availability analysis with repair



If we assume that there are two independent repair teams, then μ on the edge from 3 to 2 gets the coefficient 2 (the rate doubles).

$$M = \begin{bmatrix} -\lambda_1 & \mu & 0 \\ \lambda_1 & -\lambda_2-\mu & 2\mu \\ 0 & \lambda_2 & -2\mu \end{bmatrix}$$

Availability analysis

- None of the diagonal elements of M are 0
- By solving the system, we can get $P_i(t)$ are compute the availability as a sum of probabilities taken over all operating states
- Usually steady-state availability rather than time dependent one is of interest
- As time approaches infinity, the derivative of the right-hand side of the equation d/dt P(t) = M • P(t) vanishes and we get time-independent relationship

$$M \cdot P(\infty) = 0$$

Two-component standby system

- Using transition matrix derived earlier, we get the following system of equations

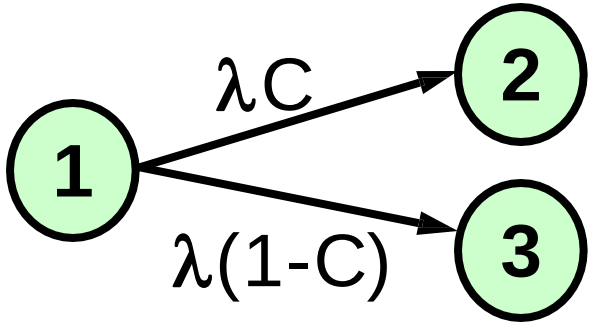
$$\begin{cases} -\lambda_1 P_1(\infty) + \mu P_2(\infty) = 0 \\ \lambda_1 P_1(\infty) - (\lambda_2 + \mu) P_2(\infty) + \mu P_3(\infty) = 0 \\ \lambda_2 P_2(\infty) - \mu P_3(\infty) = 0 \end{cases}$$

- By solving the equations, we get

$$A(\infty) \approx 1 - (\lambda/\mu)^2$$

WEEK 10
SLIDES 103-110

Safety evaluation



- The state transition equations are:

$$\frac{d}{dt}\begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix} = \begin{bmatrix} -\lambda & 0 & 0 \\ \lambda C & 0 & 0 \\ \lambda(1-C) & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

Safety evaluation

- By solving these equations, we get

$$P_1(t) = e^{-\lambda t}$$

$$P_2(t) = C(1- e^{-\lambda t})$$

$$P_3(t) = (1-C) – (1-C)e^{-\lambda t}$$

- Since the $P_i(t)$ are known, we can compute the reliability of the system as a sum of probabilities of neing the operational and fail-safe states

$$R(t) = P_1(t) + P_2(t) = C + (1-C)e^{-\lambda t}$$

- At time t=0, the safety is 1. As time approaches infinity, the safety approaches C

How to deal with cases of systems with "k out of n choices"

- Suppose we want to solve the following task:

  What is the probability that more than two engines in a 4-engine airplane will fail during a t-hour flight if the failure rate of a single engine is λ per hour?

- The probability that more than two engines fail can be expressed as:

$$P_{>2 \text{ failed}} = \binom{4}{1} P_{1 \text{ works } 3 \text{ failed}} + P_{4 \text{ failed}}$$
$$= 1 - (P_{4 \text{ work}} + \binom{4}{3} P_{3 \text{ work } 1 \text{ failed}} + \binom{4}{2} P_{2 \text{ work } 2 \text{ failed}})$$

- Only probabilities of mutually exclusive events can be summed up like this

“**k** out of n **choices**”

- “k out of n choices” can be computed as

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

- For example

$$\binom{4}{2} = \frac{4!}{(4-2)!\,2!} = 6$$

Example cont.

So, we get

$$P_{>2 \text{ failed}} = 4\, P_{1 \text{ works } 3 \text{ failed}} + P_{4 \text{ failed}}$$

where

$$P_{1 \text{ works } 3 \text{ failed}} = R\,(1-R)^3$$

$$P_{4 \text{ failed}} = (1-R)^4$$

where R is the reliability of a single engine computed as $R = e^{-\lambda t}$

Summary

- Methods for evaluating the reliability, availability and safety of a system
  - RBDs
  - Markov chains

Next lecture

- Hardware redundancy

Read chapter 4

of the text book

WEEK 11
SLIDES 111-122

HARDWARE REDUNDANCY

Techniques for fault tolerance

- Fault masking “hides” faults that occur. Do not require detecting faults, but require containment of faults (the effect of all faults should be local)
- Another approach is to first to detect, locate and contain faults, and then to recover from faults using reconfiguration

Redundancy

- hardware redundancy
  - 2nd CPU, 2nd ALU, ...
- software redundancy
  - validation test...
- information redundancy
  - error-detecting and correcting codes, ...
- time redundancy
  - repeating tasks several times, ...

Example

- FT digital filter
  - acceptance test [0 - 255]
    - SW: detect overflow
    - HW: memory for test
    - time: to execute test
  - transients: via re-execution
    - time to re-execute

Redundancy (5)

- NOTHING FOR FREE!
- costs
  - HW: components, area, power, ...
  - SW: development costs, ...
  - information: extra HW to code / decode
  - time: faster CPUs, components
- trade-off against increase in dependability

Types of redundancy

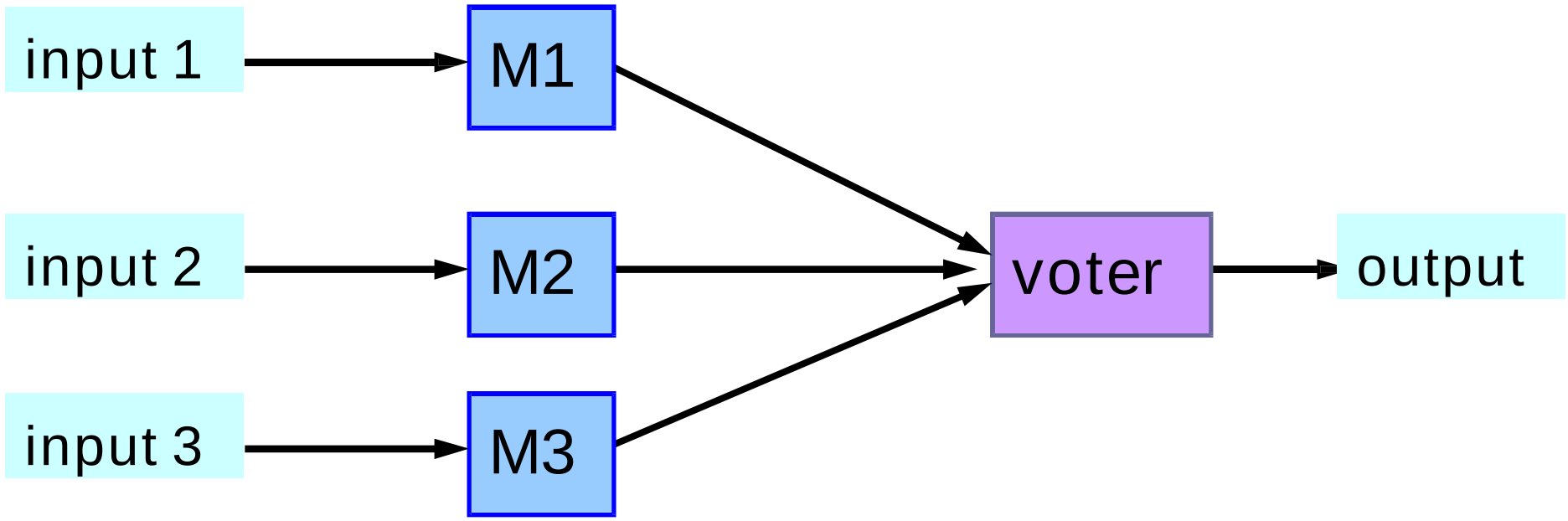
- hardware redundancy
- information redundancy
- software redundancy
- time redundancy

HW redundancy: overview

- passive redundancy techniques
  - fault masking
- active redundancy techniques
  - detection, localisation, containment, recovery
- hybrid redundancy techniques
  - static + dynamic
  - fault masking + reconfiguration

Passive HW redundancy

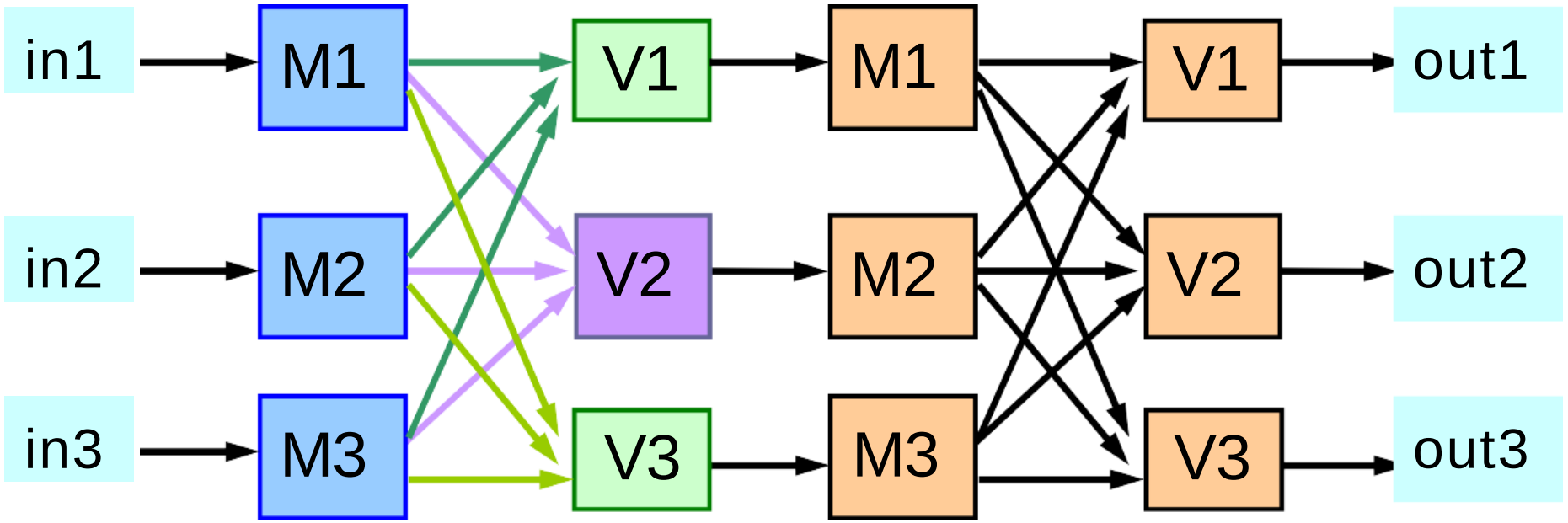
Triple Modular Redundancy (TMR)



Passive HW redundancy

- Triple Modular Redundancy (TMR)
  - 3 active components
  - fault masking by voter
- Problem: voter is a single point of failure

Passive HW redundancy



restoring organ

Passive HW redundancy

- N-modular redundancy (NMR)
  - N active components (N A)
  - N odd, for majority voting
  - tolerates $\lfloor N/2 \rfloor$ module faults

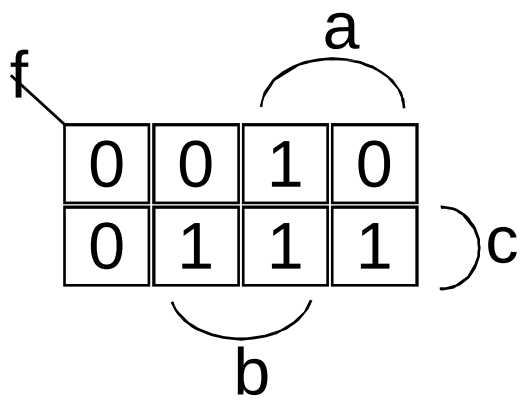
- example Apollo
  - N=5
  - 2 faults can be tolerated (masked)

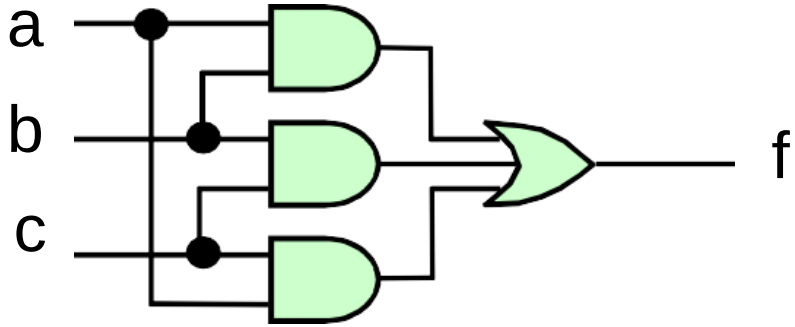
WEEK 12
SLIDES 123-130

HW voting

hardware realisation of 1-bit majority voter

$f = ab + ac + bc$





n-bit majority voter: n times 1-bit

requires 2 gate delays

SW voting

- Voting can be performed using software
- voter is software implemented by a microprocessor
- voting program can be as simple as a sequence of three comparisons, with the outcome of the vote being the value that agrees with at least on on the other two

HW vs. SW Voting

- HW: fast, but expensive
  - 32-bit voter: 128 gates and 256 flip-flops
  - 1 TMR level = 3 voters
- SW: slow, but more flexible
  - use existing CPUs

Problem with voting

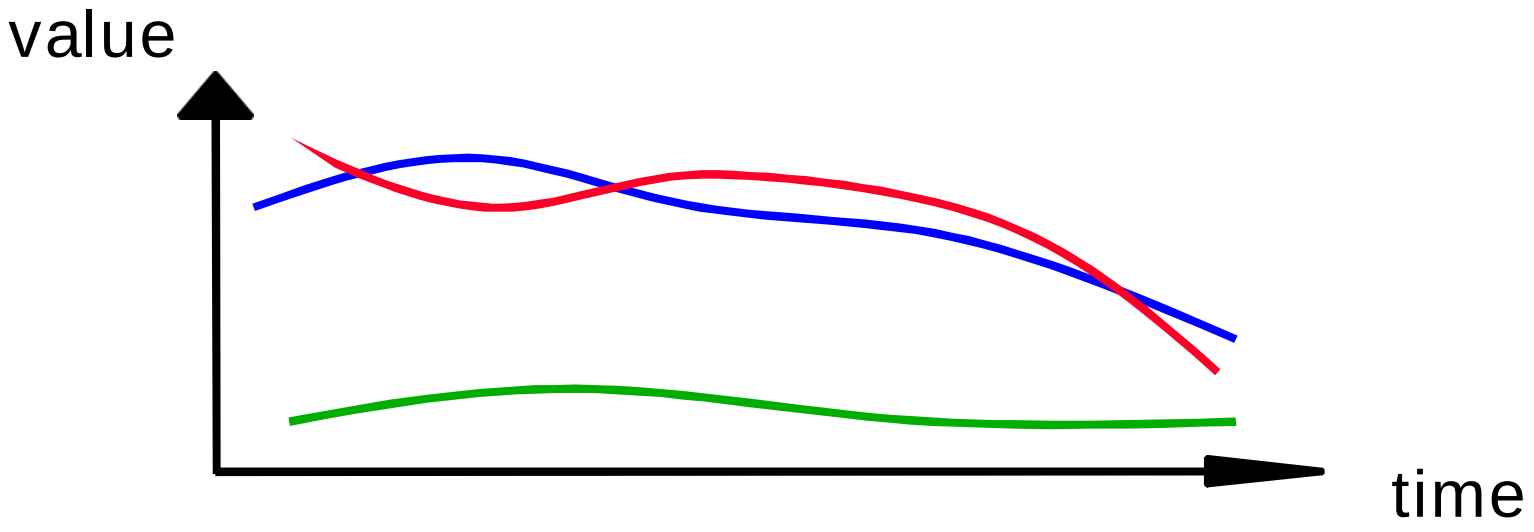
- Major problem with practical application of voting is that the three results may not completely agree
  - sensors, used in many control systems, can seldom be manufactured so that their values agree exactly
  - analog-to-digital converter can produce quantities that disagree in the least significant bits

Problems with voting

- (1) When values that disagree slightly are processed, the disagreement can grow larger
  - small difference in inputs can produce large differences in outputs
- (2) A single result must ultimately be produced
  - potential point where one failure can cause a system failure

How to cure problem 1

- Mid-value select technique
  - choose a value from the three which lies between the other two



How to cure problem 1

- Ignore the least-significant bits of data
  - disagreement which occurs only in the least-significant bits is acceptable
  - disagreement which affects the most-significant bits is not acceptable and must be corrected

WEEK 13
SLIDES 131-144

Types of HW redundancy

- static techniques (passive)
  - fault masking
- dynamic techniques (active)
  - detection, localisation, containment and recovery
- hybrid techniques
  - static + dynamic
  - fault masking + reconfiguration

Active HW redundancy

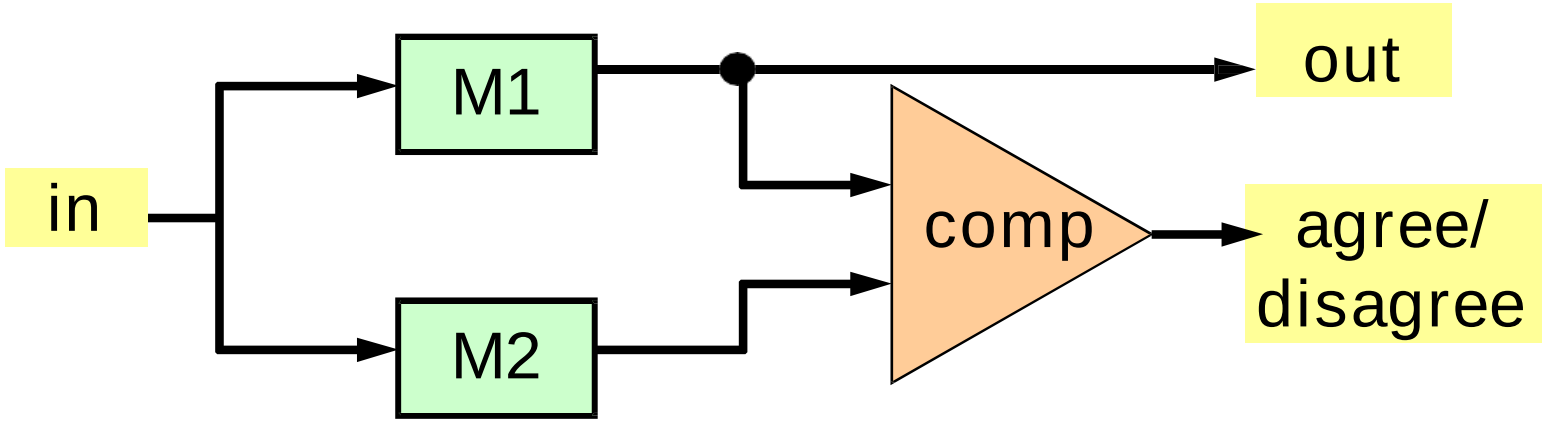
- dynamic redundancy
  - actions required for correct result
    - detection, localization, containment, recovery
    - no fault masking
  - does not attempt to prevent faults from producing errors within the system

Active HW redundancy

- most common in applications that can tolerate temporary erroneous results
  - satellite systems - preferable to have temporary failures that high degree of redundancy
- types of active redundancy:
  - duplication with comparison
  - standby sparing
  - pair-and-a-spare
  - watchdog timer

Duplication with comparison

- Two identical modules perform the same computation in parallel and their results are compared



Duplication with comparison

- The duplication concept can only detect faults, not tolerate them
  - there is no way to determine which module is faulty

Duplication with comparison

- Problems:
  - if there is a fault on input line, both modules will receive the same erroneous signal and produce the erroneous result
  - comparator may not be able to perform an exact comparison
    - synchronisation
    - no exact matching
  - comparator is a single point of failure

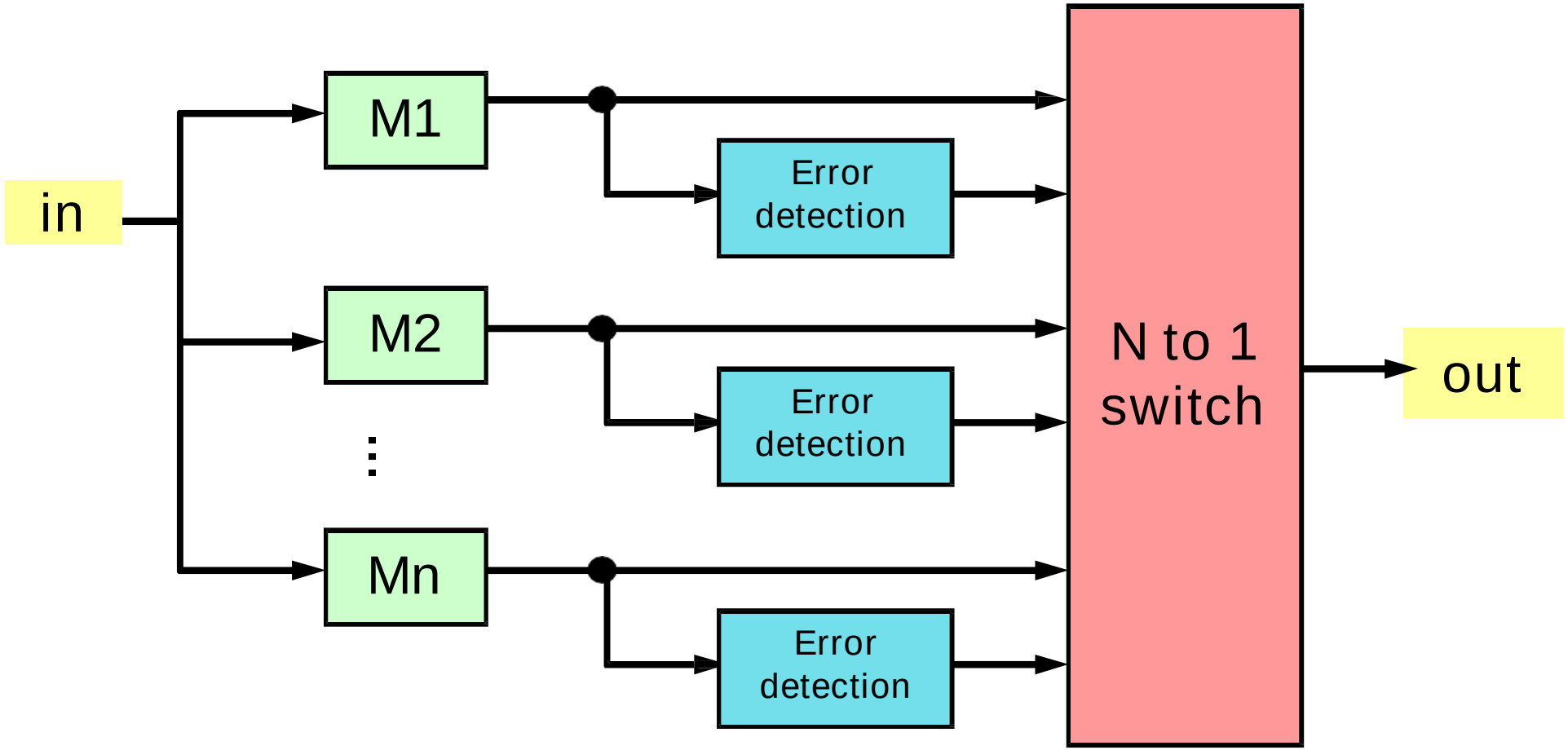
Implementation of comparator

- In hardware, a bit-by-bit comparison can be done using two-input exclusive-or gates
- In software, a comparison can be implemented a a COMPARE instruction
  - commonly found in instruction sets of almost all microprocessors

Standby sparing

- One module is operational and one or more serve as stand-bys, or spares
- error detection is used to determine when a module has become faulty
- error location is used to determine which module is faulty
- faulty module is removed from operation and replaced with a spare

Standby sparing

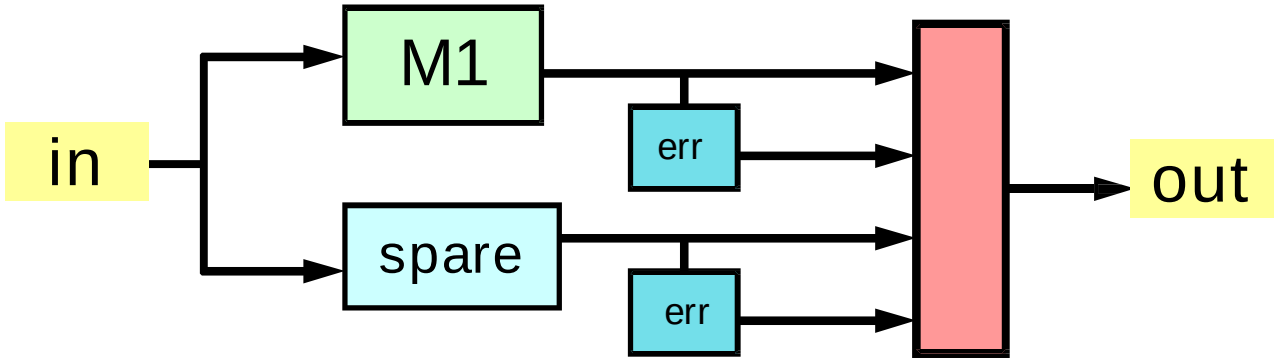


Switch

- The switch examines error reports from the error detection circuitry associated with each module
  - if the module is error-free, the selection is made using a fixed priority
  - any module with errors is eliminated from consideration
  - momentary disruption in operation occur while the reconfiguration is performed

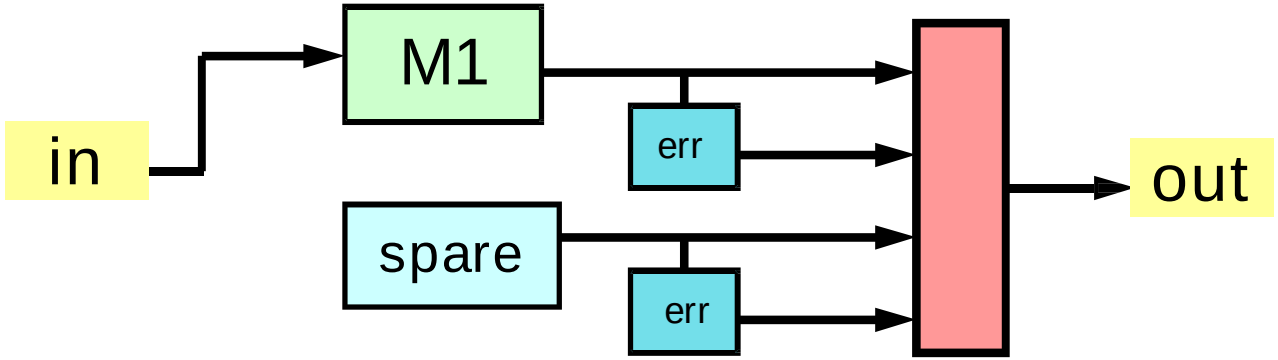
Hot standby sparing

- In hot standby sparing spares operate in synchrony with on-line module and are prepared to take over any time



Cold standby sparing

- In cold standby sparing spares are unpowered until needed to replace a faulty module



+ and - of cold standby sparing

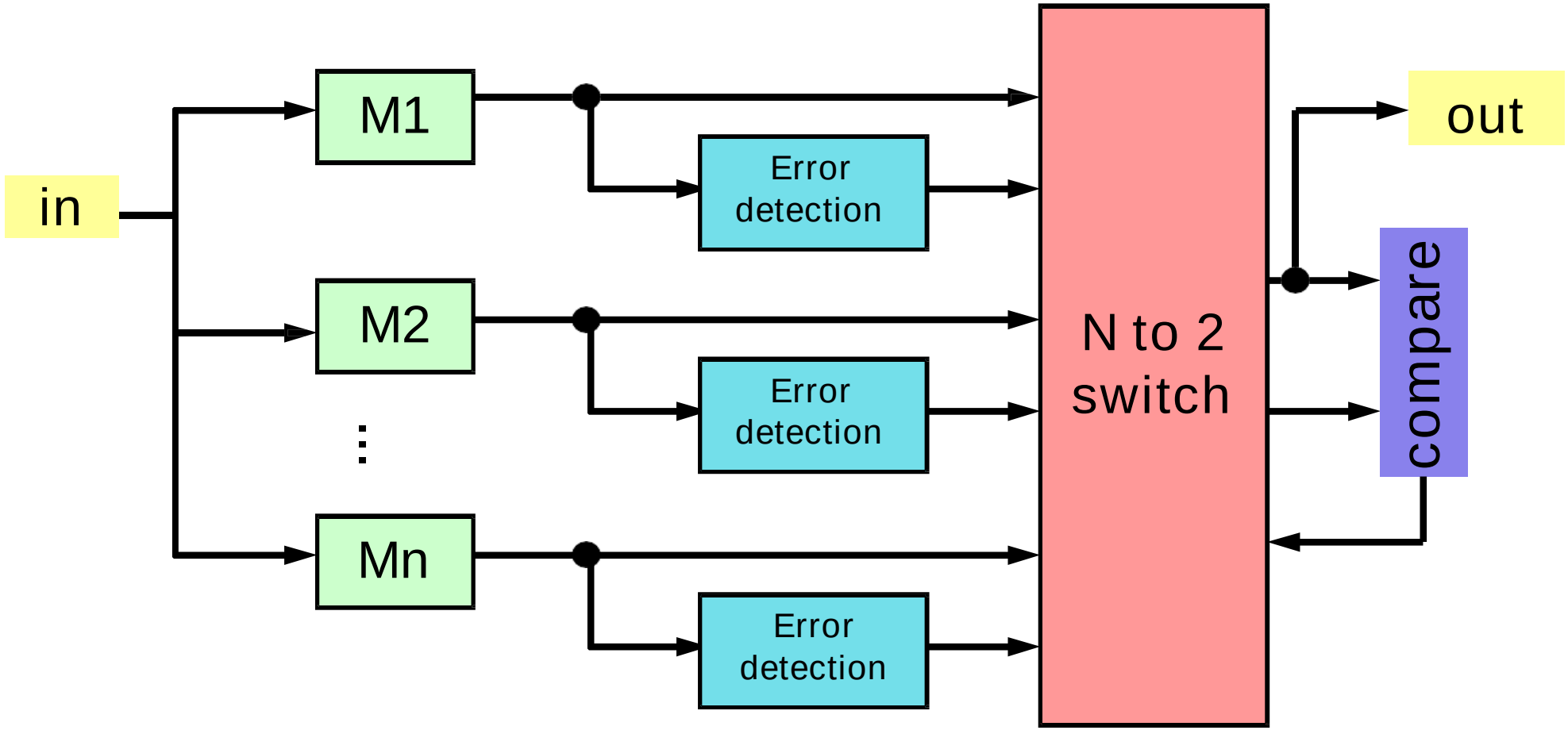
- (-) time is required to bring the module to operational state
  - time to apply power to spare and to initialize it
  - not desirable in applications requiring minimal reconfiguration time (control of chemical reactions)
- (+) spares do not consume power
  - desirable in applications where power consumption is critical (satellite)

WEEK 14
SLIDES 145-163

Pair-and-a-spare technique

- Combines standby sparing and dublication with comparison
- like standby sparing, but two instead of one modules are operated in parallel at all times
  - their results are compared to provide error detection
  - error signal initiates reconfiguration

Pair-and-a-spare technique



Pair-and-a-spare technique

- As long as two selected outputs agree, the spares are not used
- If they disagree, the switch uses error reports to locate the faulty module and to select the replacement module

Watchdog timer

- watchdog timer
  - must be reset an on a repetitive basic
  - if not reset - system is turned off (or reset)
  - detection of
    - crash
    - overload
    - infinite loop
  - frequency depends on application
    - aircraft control system - 100 msec
    - banking - 1 sec

HW redundancy: overview

- static techniques (passive)
  - fault masking
- dynamic techniques (active)
  - detection, localisation, containment, recovery
- hybrid techniques
  - static + dynamic
  - fault masking + reconfiguration

Hybrid HW redundancy

- combines
  - static redundancy
    - fault masking
  - dynamic redundancy
    - detection, location, containment and recovery
- very expensive but more FT

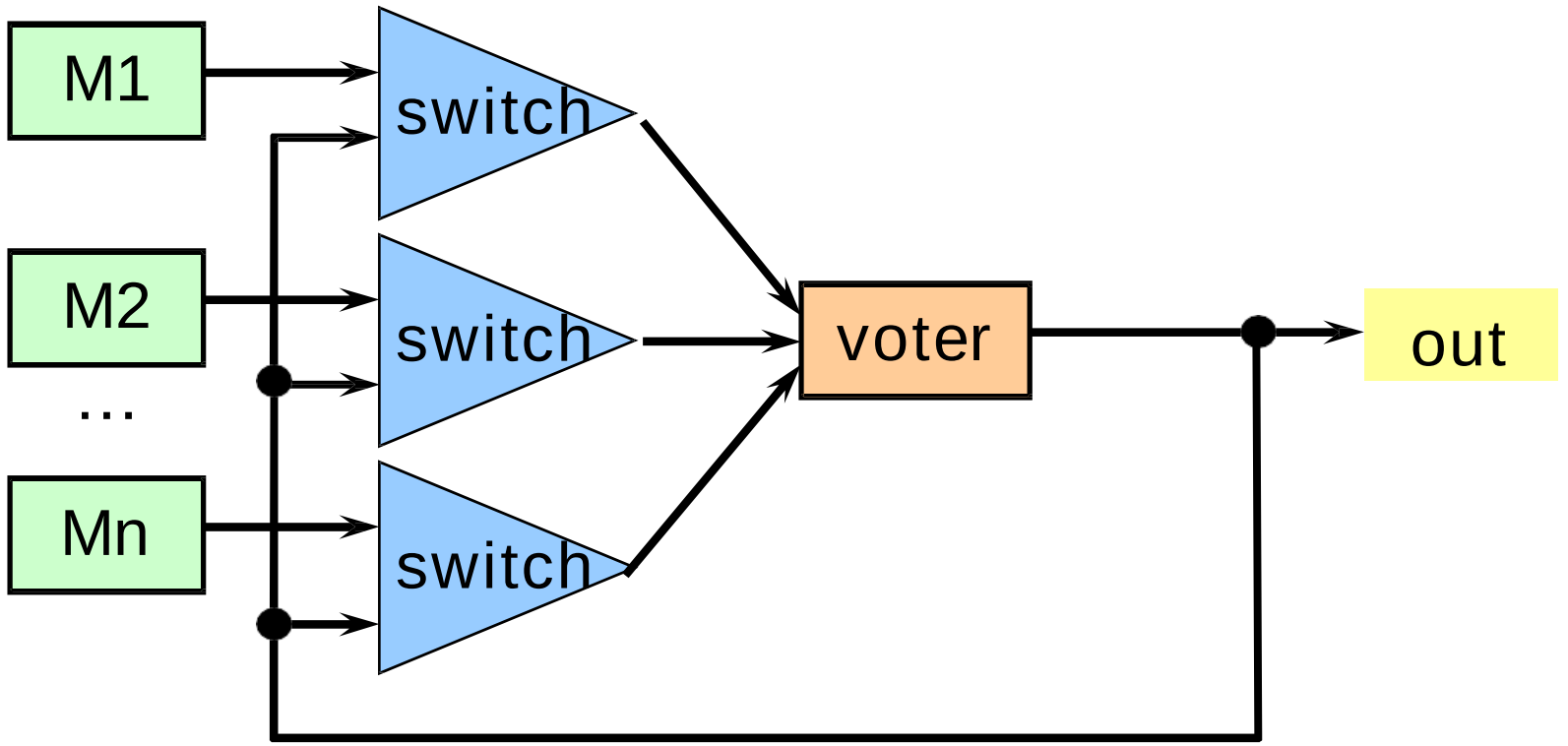
Types of hybrid redundancy

- Self-purging redundancy
- N-modular redundancy with spares
- Triple-duplex architecture

Self-purging redundancy

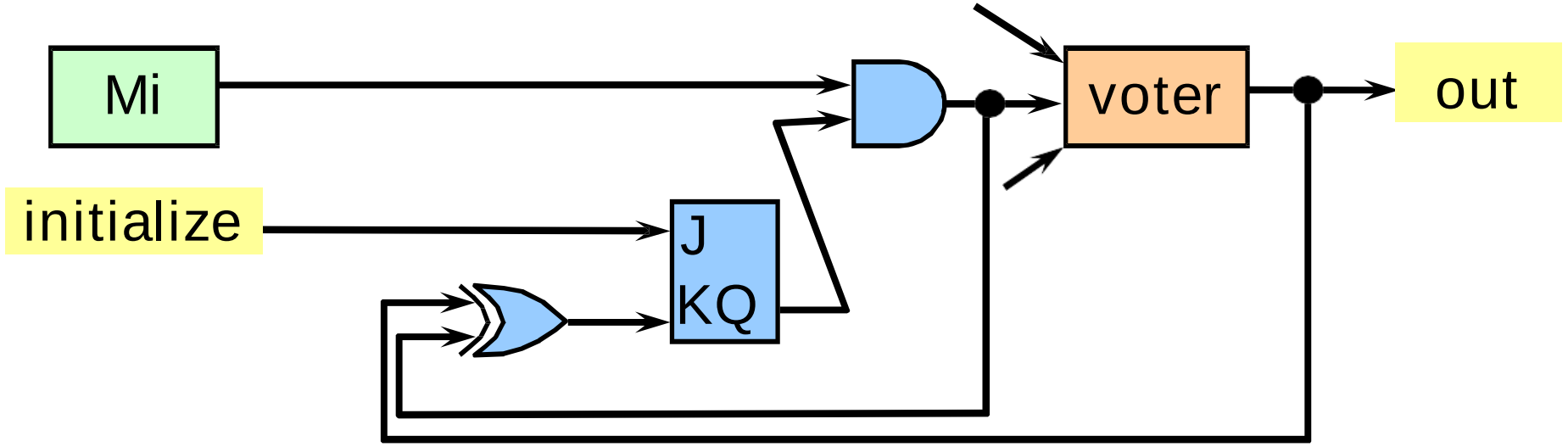
- All units are actively participate in the system
- each module has a capability to remove itself from the system if its faulty
  - very attractive feature: maintenance personnel can disable individual modules and replace them without interrupting the system

Self-purging redundancy

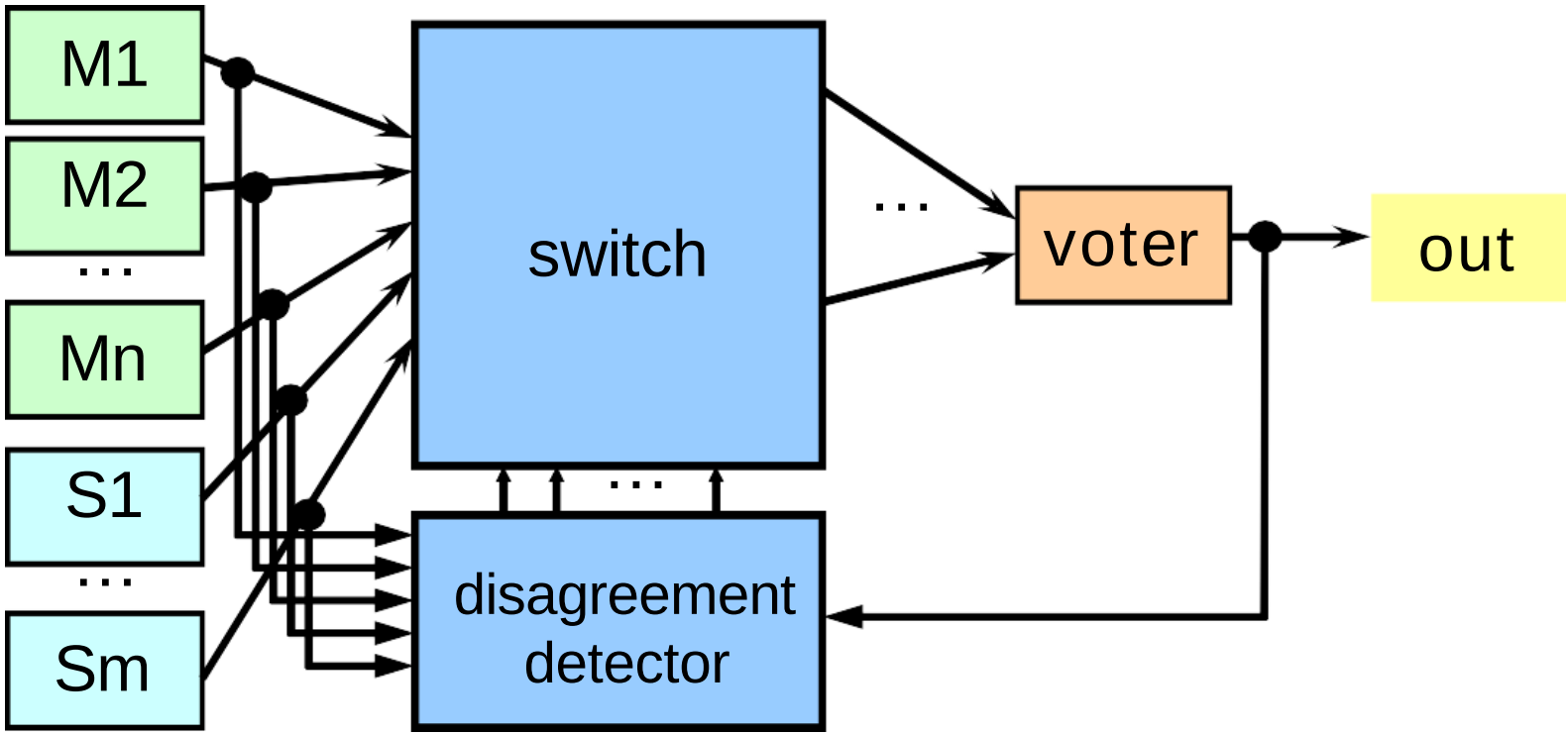


Basic structure of a switch

- Is output of a module disagrees with the output of the system, its contribution to the voter is forced to be 0 (threshold voter)



N-modular redundancy with spares



NMR with spares

- System remains in the basic NMR configuration until the disagreement vector determines a fault
- the output of the voter is compared to the individual outputs of the modules
- module which disagrees is labeled as faulty and removed from the NMR core
- spare is switched to replace it

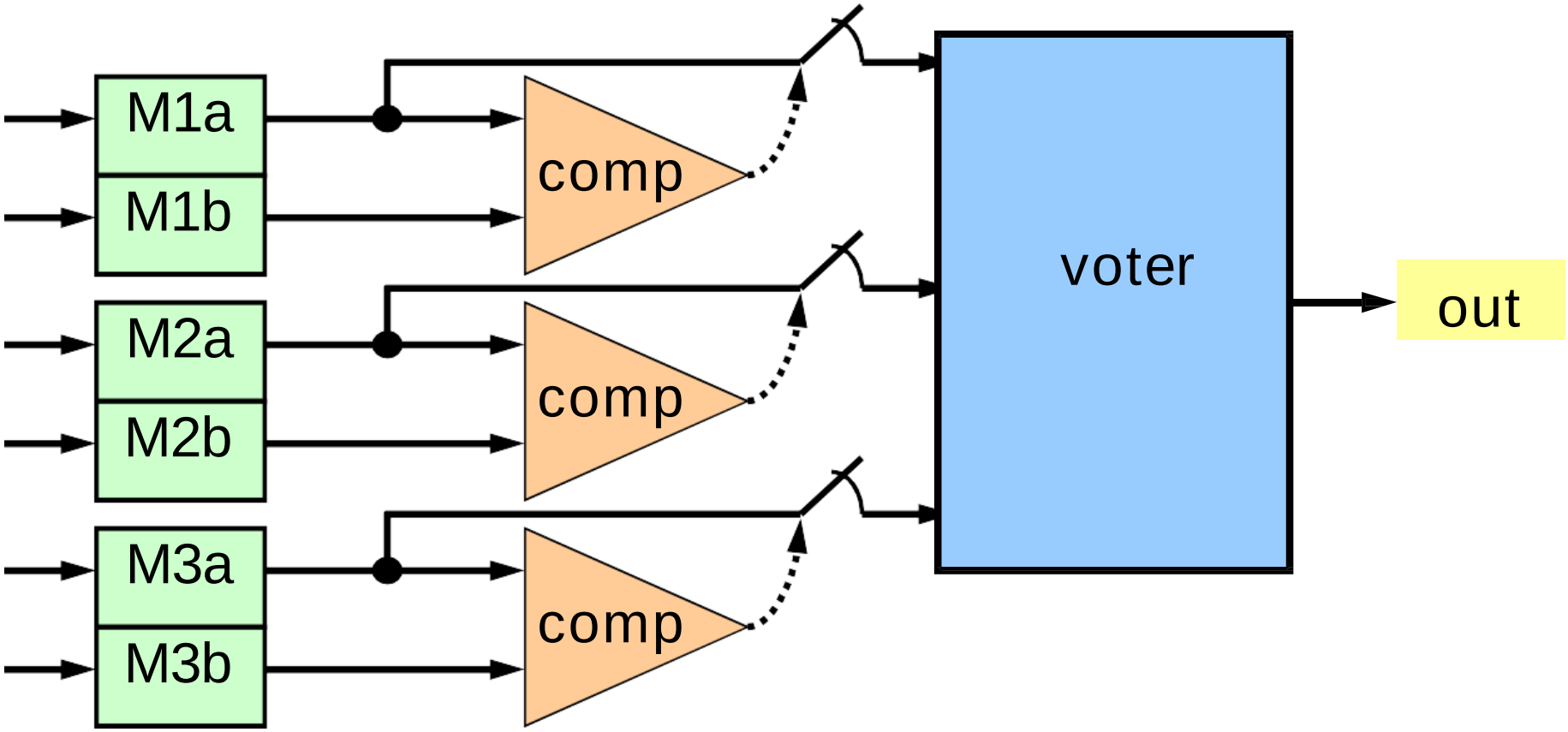
NMR with spares

- The reliability is maintained as long as the pool of spares is not exhausted
- 3-modular redundancy with 1 spare can tolerate 2 faults
- to do it in a passive approach, we would need to have 5 modules

Triple-duplex architecture

- Combines duplication with comparison and triple modular redundancy

Triple-duplex architecture



Triple-duplex architecture

- TMR allows faults to be masked
  - performance without interruption
- duplication with comparison allows faults to be detected and faulty module removed from voting
  - removal of faulty module allows to tolerate future faults
- two module faults can be tolerated

Summary

- application-dependent choice
  - critical-computation - momentary erroneous results are not acceptable
    - passive or hybrid
  - long-life, high-availability - system should be restored quickly
    - active
  - very critical applications - highest reliability
    - hybrid

Next lecture

- Information redundancy

Read chapter 5

of the text book

WEEK 15
SLIDES 164-184

INFORMATION REDUNDANCY

Information redundancy

- add information to date to tolerate faults
  - error detecting codes
  - error correcting codes
- data applications
  - communication
  - memory

Code

- Code of length n is a set of n-tuples satisfying some well-defined set of rules

- binary code uses only 0 and 1 symbols
  - binary coded decimal (BCD) code
    - uses 4 bits for each decimal digit

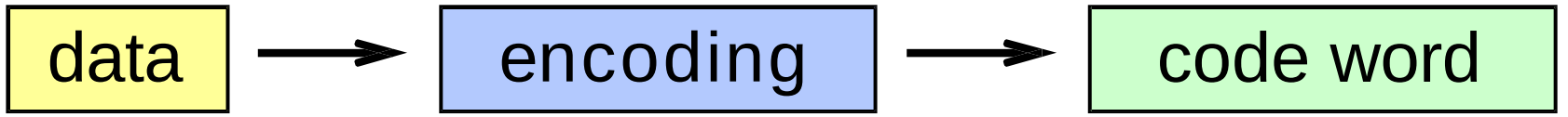
0000	0
0001	1
0010	2
...	
1001	9

Code word

- Codeword is an element of the code satisfying the rules of the code
- Word is an n-tuple not satisfying the rules of the code
- Codewords should be a subset of all possible 2^n binary tuples to make error detection/correction possible
  - BCD: 0110 valid; 1110 invalid
  - any binary code: 2013 invalid
- The number of codewords in a code C is called the size of C

Encoding/decoding

- encoding
  - transform data into code word



- decoding
  - recover data from code word

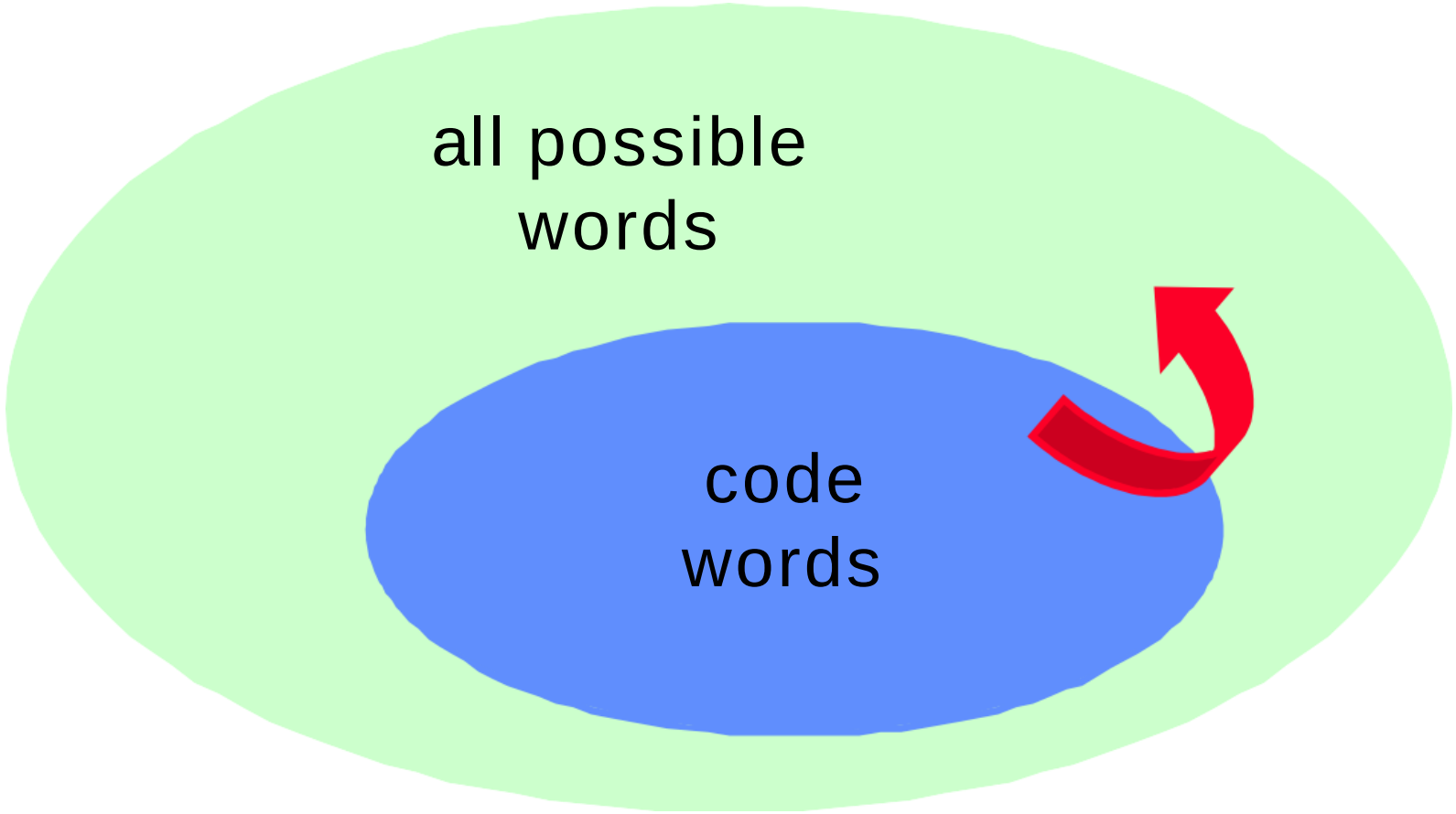
Encoding/decoding

- 2 scenario if errors affect codeword:
  - correct codeword → another codeword
  - correct codeword → word

Error detection

- We can define a code so that errors introduced in a codeword force it to lie outside the range of codewords
  - basic principle of error detection

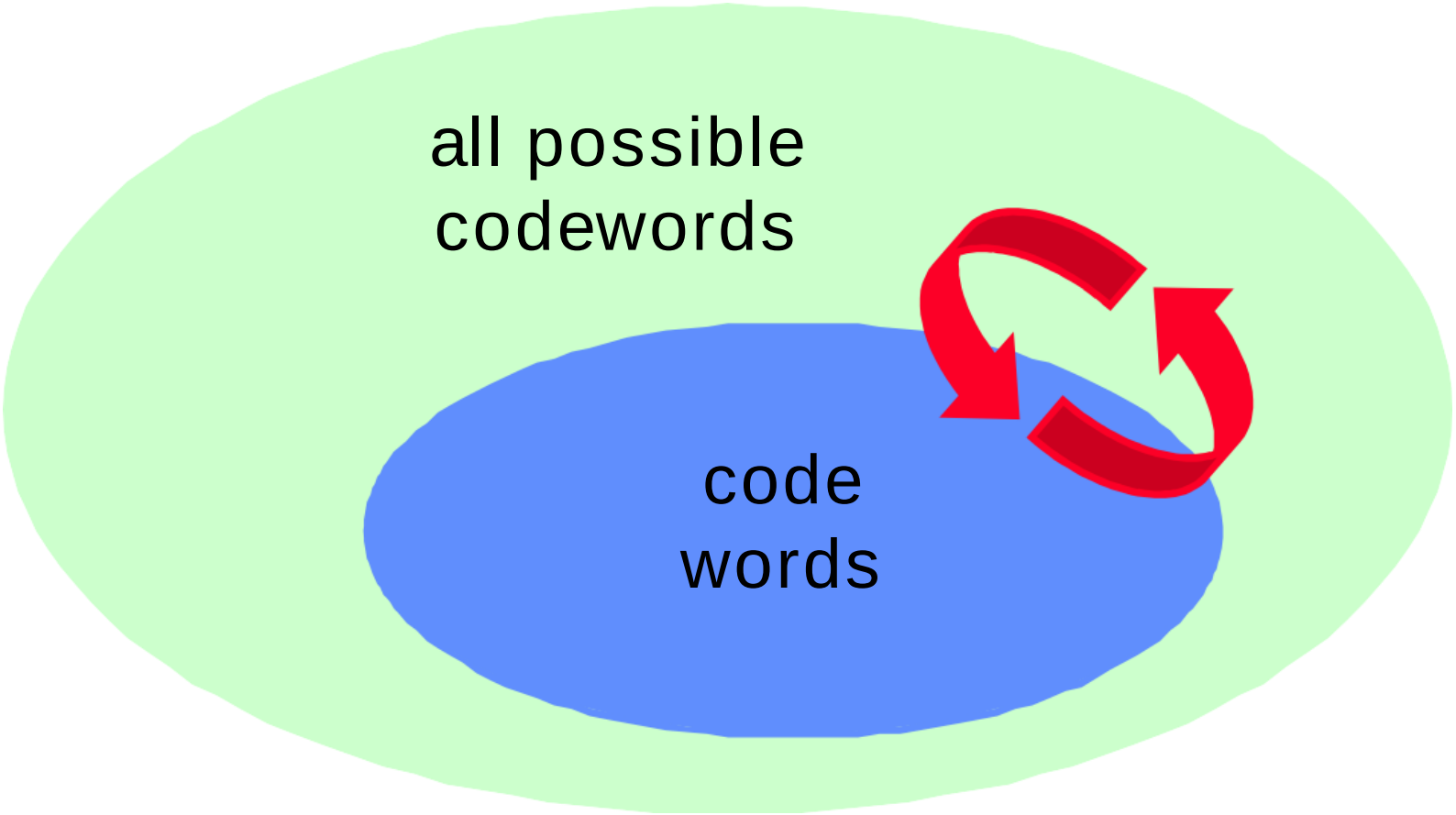
Error detection



Error correction

- We can define a code so that it is possible to determine the correct code word from the erroneous codeword
  - basic principle of error correction

Error correction



Error detecting/correcting code

- Characterized by the number of bits that can be corrected
  - double-bit detecting code can detect two single-bit errors
  - single-bit correcting code can correct one single-bit error
- Hamming distance gives a measure of error detecting/correcting capabilities of a code

Hamming distance

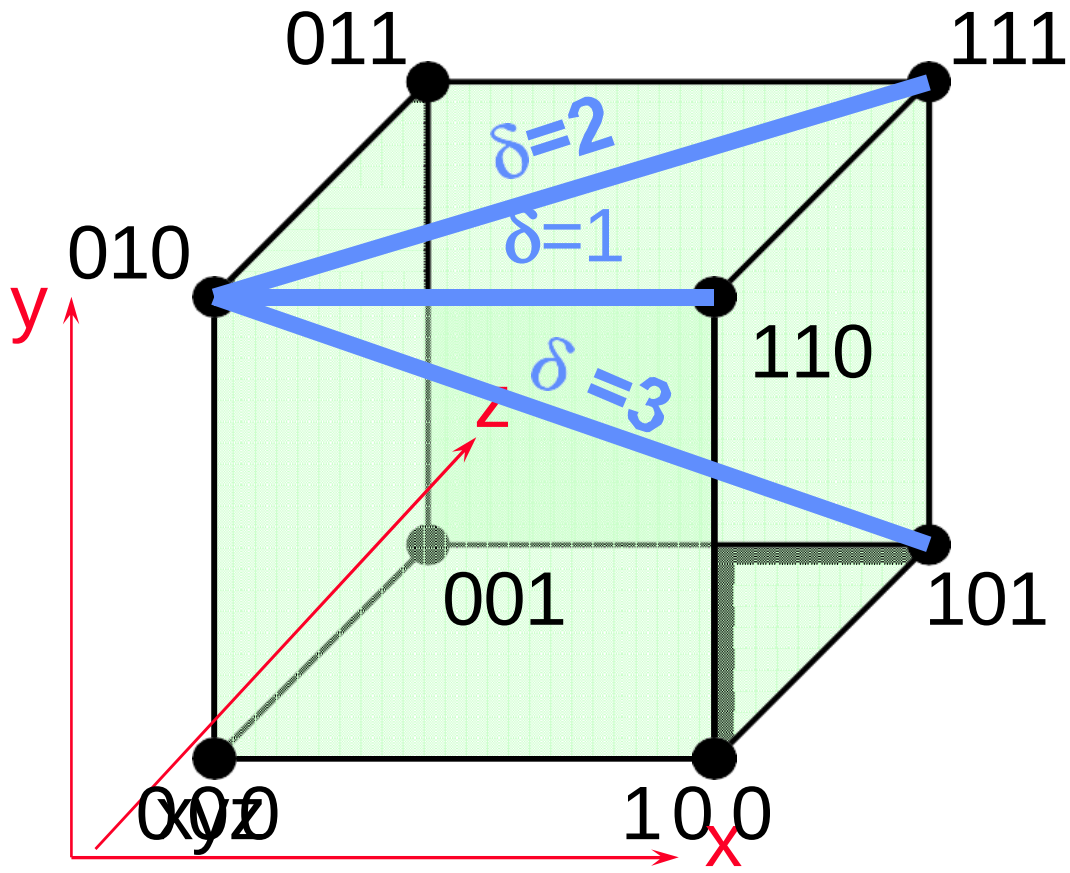
Hamming distance is the number of bit positions in which two n-tuples differ

x 0000

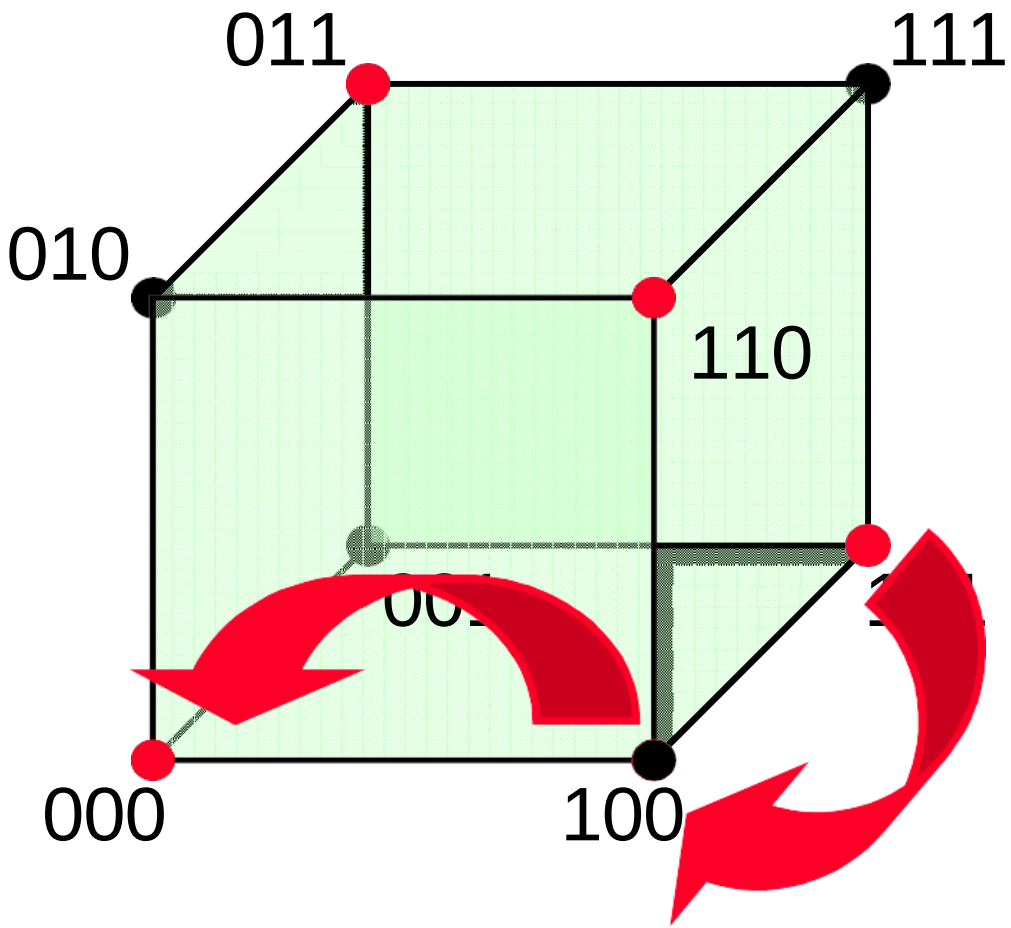
y 0101

$\delta(x,y) = 2$

3-dimensional space (3-bit words)

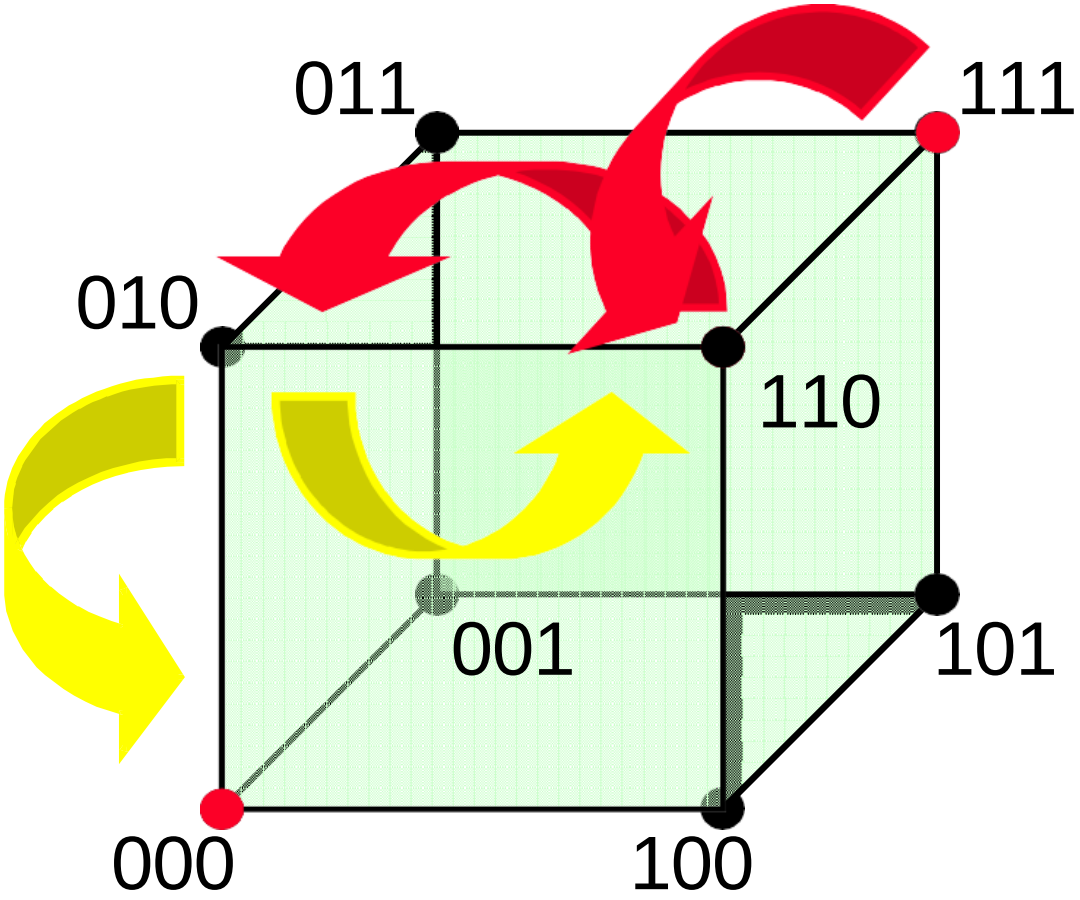


Error detection



If codewords are on distance ≥ 2, we can detect single-bit errors

Error correction



If codewords are on distance ≥ 3, we can correct single-bit errors

Code distance

code distance is the minimum Hamming distance between any two distinct codewords

$C_d = 2$ code detects all single-bit errors

code: 00, 11

invalid code words: 01 or 10

$C_d = 3$ code corrects all single-bit errors

code: 000, 111

invalid code words: 001, 010, 100, 101, 011, 110

Relation b/w code distance and capabilities of the code

A code can correct up to c bit errors and

detect up to d additional bit errors if and only if:

$$2c + d + 1 \leq C_d$$

Separable/non-separable code

- separable code
  - codeword = data + check bits
  - e.g. parity: 11011 = 1101 + 1
- non-separable code
  - codeword = data mixed with check bits
  - e.g. cyclic: 1010001 -> 1101
- decoding process is much easier for separable codes (remove check bits)

Information rate

- The ratio k/n, where
  - k is the number of data bits
  - n is the number of data + check bits

is called the information rate of the code

- Example: a code obtained by repeating data three times has the information rate 1/3

Next: Types of codes

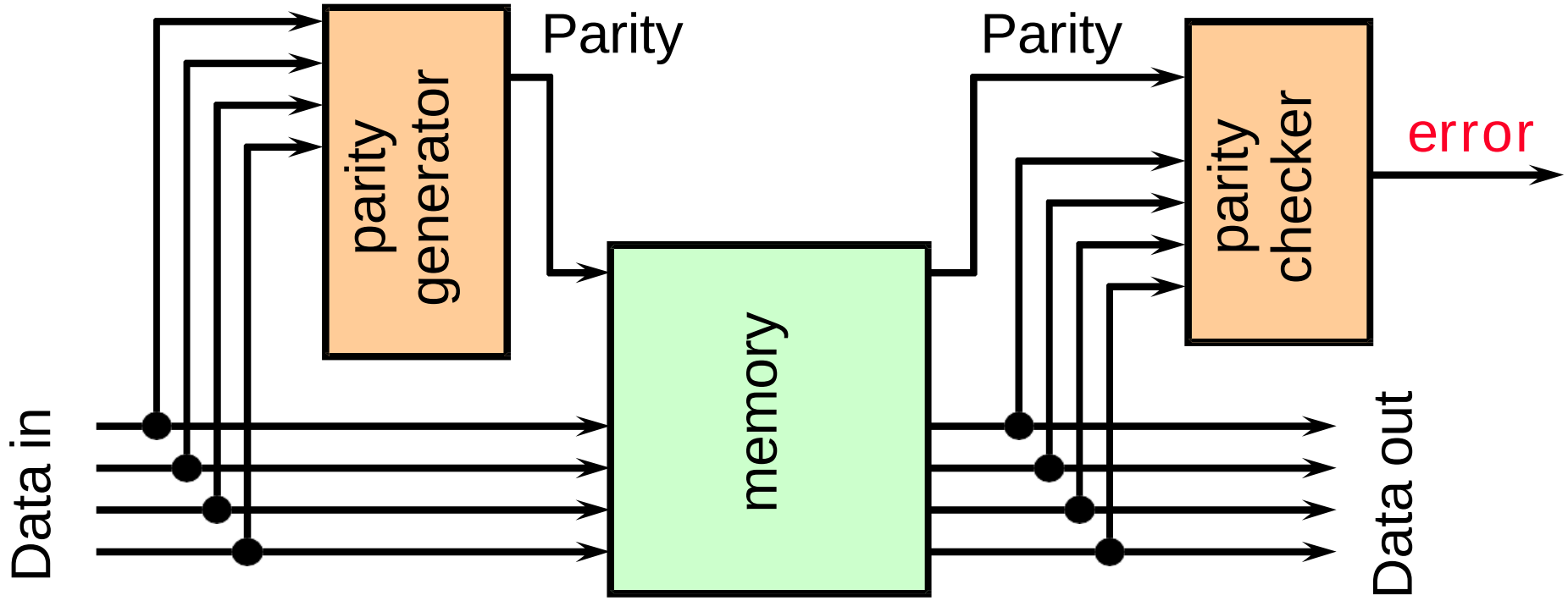
- parity codes
- linear codes
  - Hamming codes
- cyclic codes
  - CRC codes
  - Reed-Solomon codes
- unordered codes
  - m-of-n codes
  - Berger codes
- arithmetic codes
  - AN-codes
  - residue codes

WEEK 16
SLIDES 185-192

Single-bit parity code

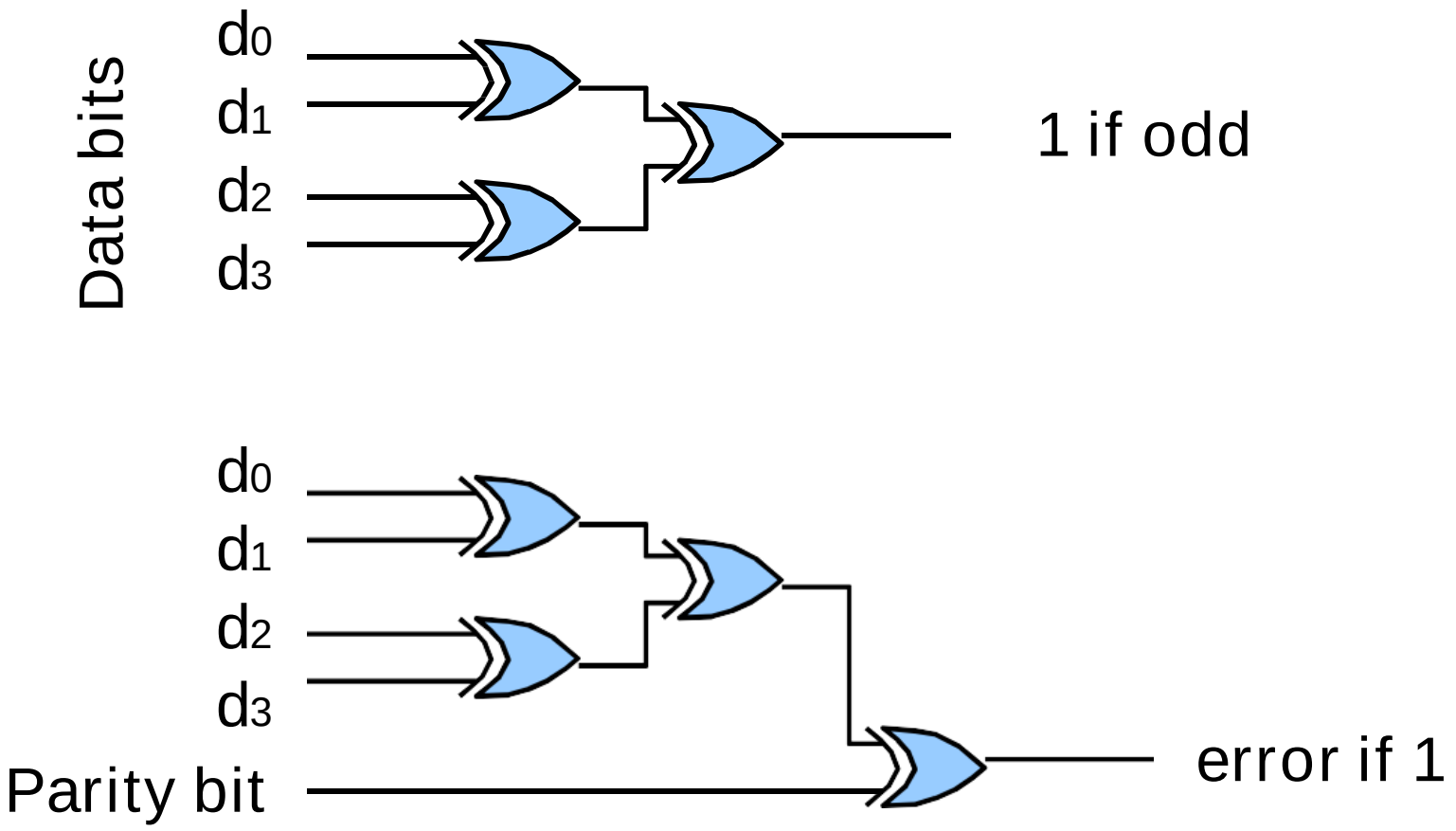
- Add an extra bit to binary word so that that resulting code word has either even or odd number of 1s
  - even parity: even # ‘1’
  - odd parity: odd # ‘1’
- single bit error detection: $C_d = 2$
- separable code
- use: bus, memory, transmission, …

Organization of memory with single-bit parity code



extra HW required (parity generator, checker, extra memory)

Parity generation and checking



Problem with single-bit parity code

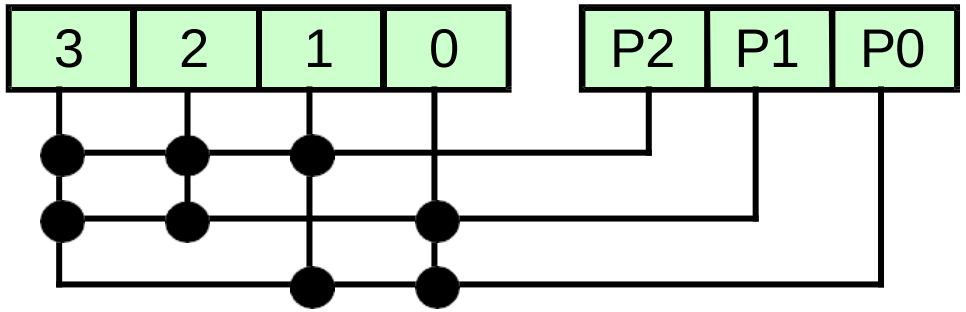
- Multiple-bit errors (even number of bits) cannot be detected
  - some of them are often very common
    - failure of the individual memory chip

Other parity codes

- The purpose is to provide additional error capability
  - bit-per-word
  - bit-per-byte
  - bit-per-multiple-chips
  - bit-per-chip
  - interlaced
  - overlapping

Overlapping parity code (Hamming code)

Overlapping parity for 4-bits of data - each data bit is assigned to multiple parity groups



Bit in error	Parity pattern
3	P2 P1 P0
2	P2 P1
1	P2 P0
0	P1 P0
P2	P2
P1	P1
P0	P0
no error	

Overlapping parity code (Hamming code)

- k data bits, c parity bits
- to have unique parity pattern per error:

$$2^c \geq k+c+1$$

k	c	redundancy
2	3	150%
4	3	75%
8	4	50%
16	5	31%
32	6	19%
64	7	11%

WEEK 17
SLIDES 193-209

Background

- A field Z_2 is the set { 0,1} together with two operations of addition and multiplication (modulo 2) satisfying a given set of properties
- A vector space V_n over a field Z_2 is a subset of Z_2^n, with two operations of addition and multiplication (modulo 2) satisfying a given set of properties
- A subspace is a subset of a vector space which is itself a vector space
- A set of vectors $\{v_0,\ldots,v_{k-1}\}$ is linearly independent if $a_0v_0 + a_1v_1 + \ldots + a_{k-1}v_{k-1} = 0$ implies $a_0 = a_1 = \ldots = a_{k-1} = 0$

Cyclic codes

- Cyclic codes are special class of linear codes
- Used in applications where burst errors can occur
  - a group of adjacent bits is affected
  - digital communication, storage devices (disks, CDs)
- Important classes of cyclic codes:
  - Cyclic redundancy check (CRC)
    - used in modems and network protocols
  - Reed-Solomon code
    - used in CD and DVD players

Cyclic code: Definition

- A linear code is called cyclic if any end-around shift of codeword produces another codeword
  - if $[c_0c_1c_2\ldots c_{n-2}c_{n-1}]$ is a codeword, then $[c_{n-1}c_0c_1c_2\ldots c_{n-2}]$, is a codeword, too
- it is convenient to think of words as polynomials rather than vectors
  - for example, a codeword $[c_0c_1\ldots c_{n-1}]$ is represented as a polynomial

$$c_0 \cdot x^0 + c_1 \cdot x^1 + \ldots + c_{n-1} \cdot x^{n-1}$$

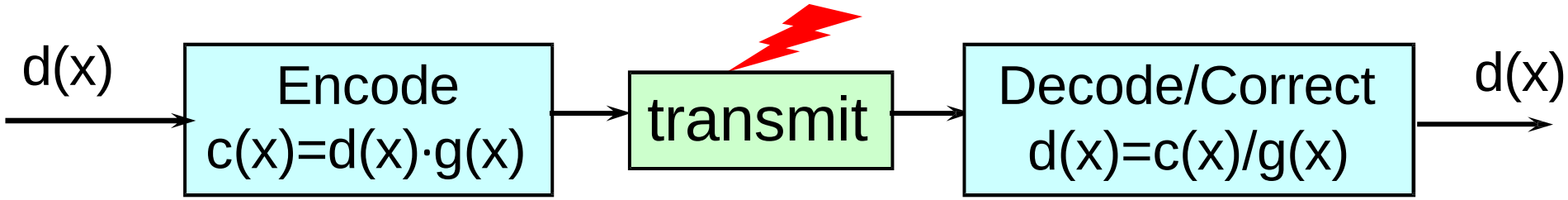
Polynomials

- Since the code is binary, the coefficients are 0 and 1
- For example, $d(x) = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$ represents the data (1011)
- We always write least significant digit on the left

Polynomials

- The degree of a polynomial equals to its highest exponent
  - e.g. the degree of $1+ x^1+ x^3$ is 3
- a cyclic code with the generator polynomial of degree (n-k) detects all burst errors affecting (n-k) bits or less
  - n is the number of bits in codeword
  - k is the number of bits in data word

Encoding/decoding



Encoding

Multiply data polynomial
by generator polynomial:

$$c(x) = d(x).g(x)$$

Calculations are performed in Galois Field GF(2):

- multiplication modulo 2 = AND operation
- addition modulo 2 = XOR operation
- in GF(2), subtraction = addition

Properties of generator polynomial

- g(x) is the generator polynomial for a linear cyclic code of length n if and only if g(x) divides $1+x^n$ without a reminder

Example of polymomial multiplication (1)

$d(x) = (1011) = x^3 + x^2 + 1$

$k = 4$

$g(x) = x^3 + x + 1$

$c(x) = d(x).g(x)$

$= (x^3 + x^2 + 1).(x^3 + x + 1)$

$= x^6 + x^4 + x^3 + x^5 + x^3 + x^2 + x^3 + x + 1$

$= x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + 1$

$= (1111111)$

$n = 7$

CRC codes

- CRC-16 and CRC-CCITT are widely used in modems and network protocols in the USA and Europe, respectively, and give adequate protection for most applications
  - the number of non-zero terms in their polynomials is small (just four)
  - LFSR required to implement encoding and decoding is simpler
- Applications that need extra protection, e.g. DD, use CRC-32

Encoding/decoding

- The encoding and decoding is done either in software, or in hardware using the usual procedure for separable cyclic codes
- To encode:
  - shift data polynomial right by deg(g(x)) bit position
  - divided it by the generator polynomial
  - the coefficients of the remainder form the check bits of the CRC codeword

Encoding/decoding

- The number check bits equals to the degree of the generator polynomial
  - an CRC detects all burst error of length less or equal than deg(g(x))
- CRC also detects many errors which are larger than deg(g(x))
  - apart from detecting all burst errors of length 16 or less, CRC-16 and CRC-CCITT are also capable to detect 99.997% of burst errors of length 17 and 99.985% burst errors of length 18

Reed-Solomon codes

- Reed-Solomon (RS) codes are a class of separable cyclic codes used to correct errors in a wide range of applications including
  - storage devices (tapes, compact disks, DVDs, bar-codes), wireless
  - communication (cellular telephones, microwave links), satellite
  - communication, digital television, high-speed modems (ADSL, xDSL).

Reed-Solomon codes

- The encoding for Reed-Solomon code is done the using the usual procedure
  - codeword is computed by shifting the data right n-k positions, dividing it by the generator polynomial and then adding the obtained reminder to the shifted data
- A key difference is that groups of m bits rather than individual bits are used as symbols of the code.
  - usually m = 8, i.e. a byte.
  - the theory behind is a field Z^m_2 of degree m over {0,1}

Encoding

- An encoder for an RS code takes k data symbols of s bits each and computes a codeword containing n symbols of m bits each
- A Reed-Solomon code can correct up to $\lfloor n-k \rfloor/2$ symbols that contain errors

Summary of cyclic codes

- Any end-around shift of a codeword produce another codeword
- code is characterized by its generator polynomial g(x), with a degree (n-k), n = bits in codeword, k = bits in data word
- detect all single errors and all multiple adjacent error affecting (n-k) bits or less

WEEK 18
SLIDES 209-222

Unordered codes

- Designed to detect unidirectional errors
- An error is unidirectional if all affected bits are changed to either $0 \rightarrow 1$ or $1 \rightarrow 0$, but not both
- Example:
  - correct codeword: 010101
  - same codeword with unidirectional errors:

110101	000101
111101	000001
111111	000000

Unidirectional error detection

- Theorem: A code C detects all unidirectional errors if and only if every pair of codewords in C is unordered
- two binary n-tuples x and y are ordered if either $x_i \leq y_i$ or $x_i \geq y_i$ for all $i \in \{1,2,...,n\}$
- Examples of ordered codewords:

$$0110 < 0111 < 1111$$

$$0110 > 0100 > 0000$$

Unidirectional error detection

- A unidirectional error always changes a word x to a word y which is either smaller or greater than x
- A unidirectional error cannot change x to a word which is not ordered with x
- Therefore, if any pair of codewords are unordered, a unidirectional error will never transform a codeword to another codeword

Berger code

- Append c check bits to k data bits

$$c = \lceil \log_2(k+1) \rceil$$

- separable code
- how to create code word:
  - count number of 1's in k data bits
  - complement resulting binary number and append it to the data bits

Example of Berger codeword

data = (0111010), k = 7

$c = \lceil \log_2(7+1) \rceil = 3$

number of 1's in (0111010) is 4 = (100)

complement of (100) is (011)

resulting codeword is (0111010011)

Berger code capability

- Berger code detects all unidirectional errors
- for the error detection capability it provides, the Berger code uses the fewest number of check bits of the available separable unordered codes

Arithmetic codes

- For checking arithmetic operations
  - before the operation, data is encoded
  - after the operation, code words are checked
- Arithmetic code is the invariant to “*” if:

$$A(b*c) = A(b) * A(c)$$

b, c - operands
A(b), A(c), A(b*c) - codes for b, c and b*c

Examples of arithmetic codes

- Two common types of arithmetic codes are
  - AN codes
  - residue codes

AN code

- AN code is formed by multiplying each data word N by some constant A
- AN codes are invariant to addition (and subtraction):

$$A(b + c) = A(b) + A(c)$$

- If no error occurred, A(b+c) is evenly divisible by A

Residue codes

- Residue codes are created by computing a residue for data and appending it to the data
- The residue is generate by dividing a data by a integer, called modulus.
- Decoding is done by simply removing the residue

Residue codes

- Residue codes are invariants with respect to addition, since

$$(b + c) \bmod m = b \bmod m + c \bmod m$$

where b and c are data words and m is modulus

- This allows us to handle residues separately from data during addition process.
- Value of the modulus determines the information rate and the error detection capability of the code

Next lecture

- Time redundancy

Read chapter 6

of the text book