

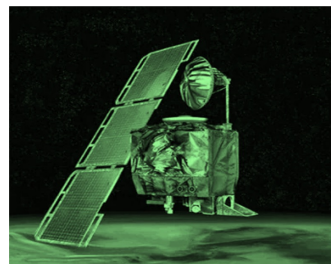
2021年5月15日早上8点20左右，中国火星探测任务“天问一号”的火星车祝融号确认着陆在火星表面。这是中国的历史性时刻，它意味着中国成为美国、俄罗斯以外，第三个实现登陆红色星球的国家。



火星着陆是探测任务中最凶险最困难的一关，也是失败率最高的阶段，整个过程被称为“恐怖七分钟”！

硬件出现故障，能更换吗？  
软件出现问题，能修补吗？  
回传数据出错，能重传吗？

全都不出故障，能实现吗？



1999年，美国发射的 Mars Climate Orbiter 号因系统故障失联。

## 容错与可靠性

### 1.可靠性:

- 系统在规定的条件下和规定的时间内,完成规定功能的能力

### 2.容错:

- 用不可靠组件，构造出可靠系统的方法
- 是获得可靠性的主要方法,也是系统设计的基本需求
























# 13. 容错与可靠性



## 本章相关的参考文献



- Gray J N, Siewiorek D P. High-availability computer systems[J]. Computer, 1991, 24(9): 39–48.
- Siewiorek D P. Architecture of fault-tolerant computers[J]. Computer, 1984, 17(8): 9–18.
- Engler D, et al. Bugs as deviant behavior: A general approach to inferring errors in systems code[C]. Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, Operating Systems Review, 2001, 35(5): 57–72.
- Swift M M, et al. Recovering device drivers[C]. Proceedings of the Sixth Symposium on Operating Systems Design and Implementation, 2004: 1–16.
- Schroeder B, Gibson G A. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?[C]. Proceedings of the Fifth USENIX Conference on File and Storage Technologies, 2007: 1–16.
- Pinheiro E, Weber W D, Barroso L A. Failure trends in a large disk drive population[C]. Proceedings of the Fifth USENIX Conference on File and Storage Technologies, 2007: 17–28.

<div>1.系统与复杂性</div> <div><div><div>HERBERT SIMON 1975</div></div><div><div>BUTLER LAMPSON 1992</div></div></div>	<div>2.构造抽象计算机系统</div> <div><div><div>BARBARA LISKOV 2008</div></div><div><div>FREDERICK BROOKS 1999</div></div><div><div>FERNANDO CORBATO 1990</div></div><div><div>DENNIS RITCHIE 1983</div></div><div><div>KENNETH THOMPSON 1983</div></div><div><div>JOHN MCCARTHY 1971</div></div><div><div>MARVIN MINSKY 1969</div></div></div>	<div>3.命名</div>		
<div>8.线程</div>	<div>7.虚拟内存</div>	<div>6.虚拟链路</div> <div><div>LESLIE LAMPOR 2013</div></div>	<div>5.虚拟化</div>	<div>4.模块化</div> <div><div>SIR TIM BERNERS-LEE 2016</div></div>
<div>9.网络设计思想</div> <div><div><div>VINTON CERF 2004</div></div><div><div>ROBERT KAHN 2004</div></div></div>	<div>10.网络分层设计</div> <div><div><div>BOB METCALFE 2022</div></div><div><div>EDSGER DIJKSTRA 1972</div></div></div>	<div>11.网络系统总体</div>	<div>12.性能</div> <div><div>JOHN COCKE 1987</div></div>	<div>13.可靠性</div> <div><div><div>JIM GRAY 1998</div></div><div><div>RICHARD W. HAMMING 1968</div></div></div>
<div><div>SILVIO MICALI 2012</div></div> <div><div>SHAFI GOLDWASSER 2012</div></div>	<div><div>ADI SHAMIR 2002</div></div> <div><div>RONALD RIVEST 2002</div></div> <div><div>LEONARD ADLEMAN 2002</div></div>	<div>16.计算机系统安全</div> <div><div><div>MARTIN HELLMAN 2015</div></div><div><div>WHITFIELD DIFFIE 2015</div></div></div>	<div>15.分布式系统</div> <div><div><div>EDGAR CODD 1981</div></div><div><div>MICHAEL STONEBRAKER 2014</div></div></div>	<div>14.原子性与一致性</div> <div><div>CHARLES BACHMAN 1973</div></div>

## 多选题 3分

设置

交叠可以减少每个请求的时延

- ☐ A 正确
- ☒ B 错误

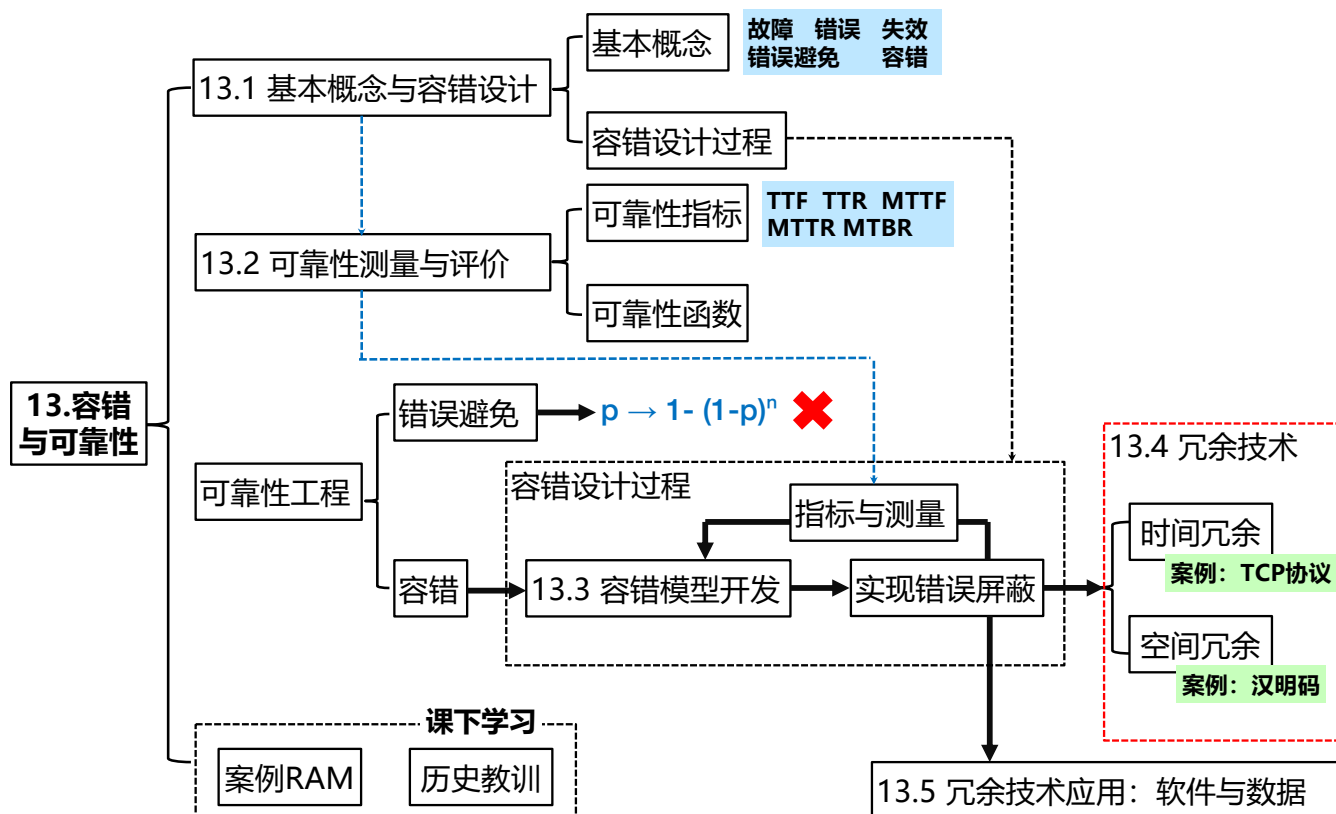
只要资源的利用率不是100%，那么队列的长度就存在上限。

- ☐ C 正确
- ☒ D 错误

假设你需要开发一个程序，程序的首要目标是尽可能减少执行时间，则你首选：

- ☒ E C或C++
- ☐ F Java或Python

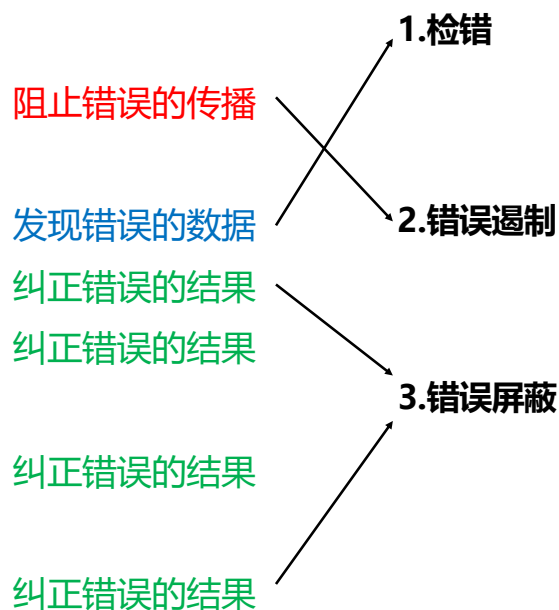
提交



## 回顾

### 出了错误怎么办？

- 第4-8章：
  - 强制模块化、虚拟内存、线程：
- 第9-11章：网络
  - 链路层检测数据错误并丢弃：
  - 端到端层可靠传输超时重发：
  - 网络层路由协议绕过坏链路：
- 系统设计技巧
  - 插值或重复丢失数据：
- 安全纵深防御
  - 设置冗余多层防护：



# 用不可靠部件构造可靠系统的3个阶段

## 1. 检错：

- 发现数据或信号的错误
- 思路：冗余

## 2. 错误遏制：

- 限制错误传播
- 思路：模块化

## 3. 错误屏蔽：

- 令系统不受错误影响，行为正常
- 思路：更多的冗余

本章课程将：

- 1.学习容错的通用原理和系统化应用
- 2.讨论实践中技术和应用的能力和限制

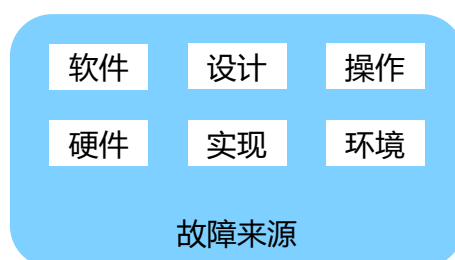
## 13.1 基本概念和容错设计

# 13.1.1 基本概念

## ① 故障 (Fault)

- **可能**导致问题的底层缺陷 (英文: bug、flaw、weakness)
- 例如:
  - 轮胎上的气泡 (可能导致车祸)
  - 软件中的悬空指针, 可能造成任意内存破坏

计算机系统的故障来自各方面原因!



## ① 故障

### 软件故障:

- 整数溢出导致符号改变
- 指针错误导致数组越界



# ① 故障

## 硬件故障：

- 逻辑门失效
- 电源故障



# ① 故障

## 设计故障：

- 内存预计不足，导致内存耗尽
- 调度各模块不匹配，导致活锁（第12章，性能）



# ① 故障

## 实现故障：

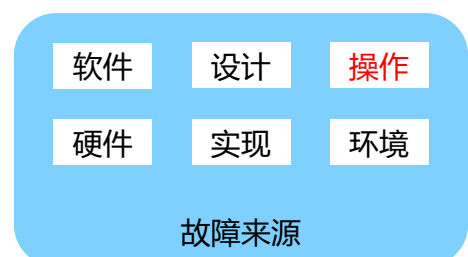
- 内存配备未遵循设计
- 程序断言被注释掉



# ① 故障

## 操作故障：

- 错误使用软件，比如 `rm -rf /*`





# ① 故障

## 环境故障：

- 雷击引发电涌，导致数据错误



# ② 错误

故障会立即或在未来产生不一致的**状态**：错误（Error）

## 错误（Error）

- 违反了设计规范中的断言或不变量的程序状态或数据值

错误相比缺陷，变得可以进行观察和测量。

但在没有断言或不变量的情况下，  
查找错误也通常没有通用方法！

## ③ 失效

如错误没有被检测和消除，模块可能产生意外行为或结果：失效

**失效 (failure)**

- 在模块接口上未产生预期结果

**组件的失效是系统的故障！**

## ④ 容错

**容错 (Fault Tolerance)**

- 发现故障和组件失效，并采取有效应对
- 应对机制
  - 在组件层面上：进行错误遏制(Error Containment)
  - 错误所在的模块是错误遏制的最小单位。

## ④ 容错

在接口层面上：

1. 屏蔽错误
2. 报告错误：快速失败 (fail-fast) 设计 ( $\neq$  failure)
3. 停止运行：失败停止 (fail-stop) 设计
  - 问题：在异步系统中，如何判断stop?
4. 无反应：（非正式称谓：crash或fail-thud = failure）

## 13.1.2 容错设计过程



**错误避免 (Fault Avoidance)：使用可靠部件搭建系统**

- 难度大，不现实（较大系统）
  - 部件故障率 $p \rightarrow$  系统故障率 $1 - (1-p)^n$

**容错：使用不可靠部件搭建可靠系统**

## 13.1.2 容错设计过程

### 容错设计过程

#### - 步骤

#### 系统开发时：

1. 开发容错模型，使用模块化遏制高风险错误产生的危害
2. 设计实现错误屏蔽机制，将机制更新到容错模型
3. 测量可靠性/错误率
4. 根据测量结果进行迭代，直到达到可接受水平

#### 系统运行中：

1. 观察并分析
2. 改进并迭代

## 13.1.2 容错设计过程

### 涉及的技术与原则

#### 风险管理方法

故障识别

风险计算

成本/收益计算

错误率/可靠性测量

检错技术

纠错技术

显式原则

迭代设计原则

安全边界原则

不断挖掘原则

彻底简化原则

#### 安全网框架

## 13.2 可靠性测量与评价

### 13.2.1 可靠性指标

#### 测量指标

- 失效时间 (Time to Failure) : TTF
- 修复时间 (Time to Repair) : TTR



#### 评价指标

- 平均失效时间:  $MTTF = \frac{1}{N} \sum_{i=1}^N TTF_i$
- 平均修复时间:  $MTTR = \frac{1}{N} \sum_{i=1}^N TTR_i$
- 平均故障间时间:  $MTBF = MTTR + MTTF$
- 可用性:  $Availibility = \frac{\text{运行时间}}{\text{应运行时间}} = \frac{\sum TTF}{\sum (TTF + TTR)} = \frac{MTTF}{MTBF}$

# 如何测量MTTF?

## 为什么要测量MTTF?

- 评估组件的可靠性，是否达成系统的设计目标
- 预测组件未来行为（需要有统计模型做支持）

## 如何测量MTTF?

- （仅考虑替换，不考虑维修）
- 测量若干组件的TTF，取其平均值

## [自学]各态历经性 (ergodic) 与系综平均 (Ensemble Average)

# 如何测量MTTF?

思考：以下硬盘的MTTF是怎么测量得到的？

## Toshiba MG06 Series Hard Disk Drives

- 10 TB, 8 TB, and 6 TB capacity models
- Industry Standard 3.5-inch 26.1 mm Height Form Factor
- 7,200 rpm Performance
- MTTF 2.5M hours **≈ 100,000 天 ≈ 270年！**

WHERE TO BUY

显然，不是测量值！  
而是MTTF的预测值！

# 如何预测MTTF?

## 统计方法

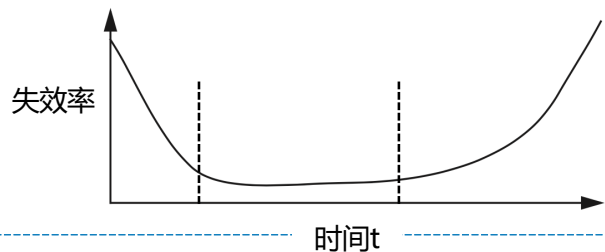
- 假设失效是时间无关的(memoryless)

在较短时间 $t$ 内, 运行 $n$ 个组件, 统计失效个数 $x$ , 进行计算

$$\frac{n \cdot t}{\text{MTTF}} \approx x \quad \Rightarrow \quad \text{MTTF} \approx \frac{n \cdot t}{x}$$

- 但有些组件的失效不是时间无关的:

- 夭折现象 (infant mortality)
- 耗竭现象 (burn out)



# 假如组件失效不是时间独立的

## 基于模型的统计方法

1. 基于经验或测量, 获得该组件故障率的概率密度函数模型
2. 假设各组件之间的故障是相互独立的, 进行一组组件测试
3. 使用测得的值进行回归分析

## 结合工程的统计方法

- 加速老化 (Accelerated Aging)
  - 需要建立老化与时间的换算模型

# 可用性指标

## 可用性指标的极简方法：N-nines 可用性

- 3-nines:  $\frac{MTTF}{MTBF} = 99.9\%$       1.5 分钟/天
- 5-nines: 99.999%      5 分/年
- 7-nines: 99.99999%      7 秒/年
- 不能体现MTTF
  - 1秒/天  $\hat{=}$  30秒/月
- 在什么情况下适合?
  - 非连续的服务，例如网络授时服务

## 13.2.2 可靠性计算

### 定义

1. 可靠性函数:  $R(t) = \Pr(\text{从开始到}t\text{无失效})$
2. 无条件故障率:  $f(t) = \Pr(\text{在}t\text{到}dt\text{间失效})$

### 平均无故障时间和可靠性计算

- $MTTF = \int_0^{\infty} t \cdot f(t) dt$ : 如果 $f(t)$ 准确，与实际测量的MTTF相等
- 当失效时间无关时（可能是函数的一部分）
  - $MTTF = 1/E$ ， $E$ 为错误率
  - $R(t) = e^{-(t/MTTF)}$
- 当失效时间相关时，无简单方法



# 13.2.2 可靠性计算

## 极简方法：nσ法

- 针对生产线产品的某个参数进行控制
- 假设参数符合正态分布，求其标准差σ
- nσ 表示均值距离错误值可以到n个标准差
- 例：4.5σ时，错误率  $\approx 3.4 \times 10^{-6}$
- 对于质量管理者，如果达不到n σ，应该做什么？

n	$p = F(\mu + n\sigma) - F(\mu - n\sigma)$	i.e. $1 - p$
1	0.682 689 492 137	0.317 310 507 863
2	0.954 499 736 104	<u>0.045 500 263 896</u>
3	0.997 300 203 937	<u>0.002 699 796 063</u>
4	0.999 936 657 516	<u>0.000 063 342 484</u>
5	0.999 999 426 697	0.000 000 573 303
6	0.999 999 998 027	0.000 000 001 973

该方法适用于生产线，不适用于安装运行的组件

## 6σ方法：

- 4.5σ：产品 偏离 生产线均值
- 1.5σ：生产线均值 偏离 规格

# 13.3 容错模型开发

# 1. 响应分类

## 对于错误可能采取的响应方式

- 无响应 (Do nothing) : 任由错误发生
- 快速失败 (fail-fast) : 发现错误立即报告
- 失败安全 (fail-safe) : 发现错误采取安全方式停止
- 软失败 (fail-soft) : 发现错误, 仍可部分正常工作
- 屏蔽错误 (Mask the error) : 发现错误并纠正错误

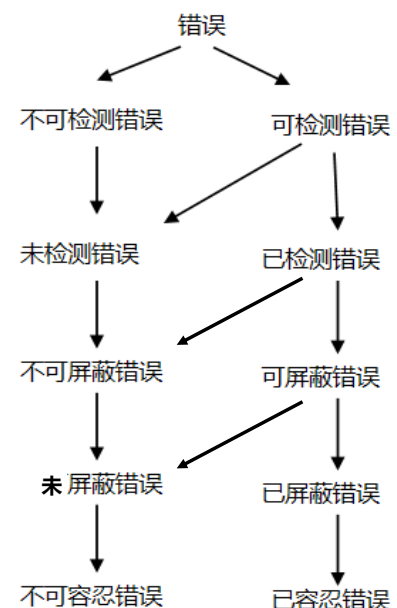
# 2. 错误分类

## 错误的分类

可检测错误 → 已检测错误 →  
→ 可屏蔽错误 → 已容忍错误

未容忍错误可能导致失效

## 错误分类用于开发容错模型



## 3. 容错模型

### 基于错误分类开发容错模型

- 1.分析系统，将可能的错误事件分类到可检测和不可检测
  - 此时所有错误标记为未容忍错误
  - 如果不可检测错误发生率不可忽略，更改系统设计
- 2.对于所有可检测错误，实现检测规程(procedure)
  - 将所能检测到该错误的模块标记为fail-fast
- 3.对于所有可检测错误，尝试设计屏蔽机制
  - 如果可行则将此错误标记为可屏蔽错误
- 4.对于可屏蔽错误，评估发生率、失效代价、屏蔽代价
  - 根据评估结果决定是否实施屏蔽，如实施则划分其为已容忍错误

没有问题  
了吗?

## 问题

1. 错误事件列表完备吗？概率估计真实吗？
  - 现实世界问题，必然包含主观的、经验的判断
2. 用于检测和屏蔽错误的算法、规程、实现等是完备和正确的吗？
  - 抽象问题，一般有客观的答案

### 可靠性设计者应对质疑的基本逻辑：

- 是质疑算法正确性？
- 还是质疑失效模型？

# 容错模型的迭代

## 迭代

- 前提：首先建立模型
- 时机：当错误发生率、类型超出估计时
- 观察：审查模型，发现模型可以改进的地方
- 行动：改进设计

## 用迭代应对技术进步

- NFS：从局域网到互联网，重新修改了容错模型
- RAM容错模型（自学内容）

## 13.4 容错技术：冗余

# 冗余：从模拟到数字

## 模拟系统的容错

- 保留安全边界

## 数字系统的容错

- 瞬时错误：重试
  - 时间上的冗余
- 永久错误：另一个相同的组件提供正确结果
  - 空间上的冗余

## 13.4.1 增量冗余：编码

### 用汉明距离 (Hamming distance) 检错与纠错

- 汉明距离：两个比特串对应位置不同字符的数量
  - 例：0011与1100的汉明距离为4
- 如两个编码汉明距离为 $d$ ，则检错能力 $d-1$ 位，纠错能力 $\lfloor d-1/2 \rfloor$ 位
- 例：
  - 奇偶校验 $n+1$ 可检1位错。（思考：汉明距离为多少？）
  - $4+3$ 可纠1位错。所以其所有编码之间的汉明距离一定  $\geq$  \_\_\_\_\_？

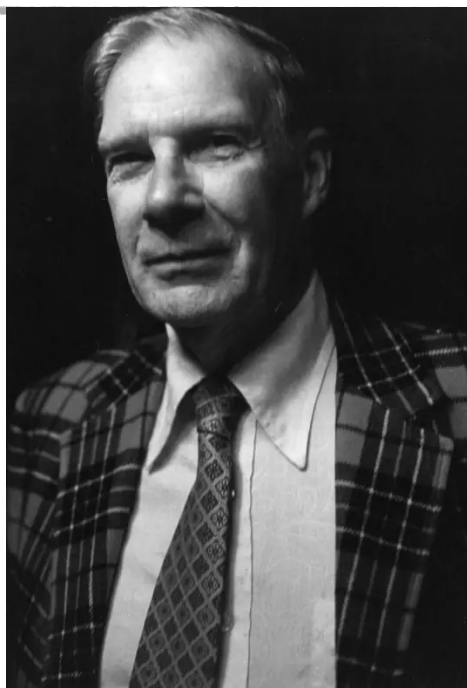
37. 若甲向乙发送数据时采用 CRC 校验，生成多项式为  $G(x) = x^4 + x + 1$ （即  $G = 10011$ ），则乙接收到下列比特串时，可以断定其在传输过程中未发生错误的是（ ）。

- A. 1 0111 0000      B. 1 0111 0100      C. 1 0111 1000      D. 1 0111 1100

# Richard Wesley Hamming

1968年图灵奖得主

贡献：数值方法，自动编码系统，  
错误检测和纠错码



## 4+3可纠1位错，如何设计？

**要求：3个冗余位，使汉明距离为3，设计一种编码方式**

- 思路：
- 对于任一编码，用3个冗余位建立3个布尔算式，保证如下性质：
  - 3个算式的和依次为000表示无错，n则表示第n位错。
- 因001表示第1位错，011表示第3位错，101表示第5位错，111表示第7位错 → 算式3  $P_1 + P_3 + P_5 + P_7 = 0$ ：这些位错 iff 算式3为1
- 因010表示第2位错，011表示第3位错，110表示第6位错，111表示第7位错 → 算式2  $P_2 + P_3 + P_6 + P_7 = 0$ ：这些位错 iff 算式2为1
- 同理：  $P_4 + P_5 + P_6 + P_7 = 0$ ：这些位错 iff 算式1为1

# 应对丢失 (erasure)

## 丢位：

- 如果知道位置，奇偶校验可恢复1位，4+3可恢复3位
- 称为纠删码 (Erasure Code)

## 为什么可以？

$$\begin{cases} P1+P3+P5+P7 = 0 \\ P2+P3+P6+P7 = 0 \\ P4+P5+P6+P7 = 0 \end{cases}$$

## 同理，丢包：

- 如果知道序号，那么奇偶校验包可恢复1个包，4+3可恢复3个包

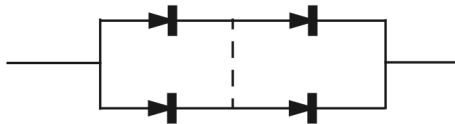
# 收益

1. 广播场景有用
2. 控制时延有效
3. 超远通信必须
4. 长期存储适用

## 13.4.2 规模冗余：复制

**复制 (replication)：**使用同一组件的多个副本 (replicas)

- 设计出错时的自动替换机制
  - 例：4组件超级二极管（香农&摩尔 50s）



思考：这个设计的不足

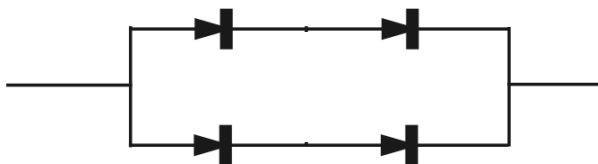
### 填空题 2分

设置

此题未设置答案，请点击右侧设置按钮

超级二极管如图所示。假设二极管仅会发生短路故障。

将二极管短路看做以 $\Delta t$ 为单位的离散事件。假设二极管在 $\Delta t$ 时间内短路的概率为0.01%且不随时间改变。4个二极管的短路概率互相独立。那么该超级二极管在 $\Delta t$ 时间内出现短路的概率约为 [填空1]。  
在 $2\Delta t$ 时间内出现短路的概率约为 [填空2]。（选做1空）



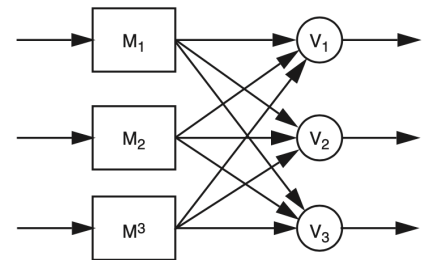
作答



## 13.4.3 多模块冗余：表决

### N个相同的组件+表决器

- N模块冗余 (N-modular redundancy, NMR)
- 称为超级模块 (supermodule)
- 表决器设计规则
  - 失败投票 (fail-vote)：服从多数意见



问题：超级模块一定比单一模块可靠吗？

#### 单选题 1分

设置

为了提高飞机的可靠性，设计师决定使用3个引擎的设计方案。但由于引擎的重量很大，使得3引擎飞机必须有2个引擎同时工作方可正常飞行。

假设单个引擎的MTTF=6000小时。3个引擎的故障是相互独立的。

问题：3引擎飞机的MTTF高于单引擎飞机吗？你选择哪个方案（慎重考虑）？

- ☐ A 高于单引擎飞机。选择3引擎飞机。
- ☒ B 低于单引擎飞机。选择3引擎飞机。
- ☐ C 高于单引擎飞机。选择单引擎飞机。
- ☐ D 低于单引擎飞机。选择单引擎飞机。

提示：3引擎同时运转，出现第1个引擎失败的MTTF约为2000小时。  
第1个引擎失败之后，2引擎继续运转……

提交

## 13.4.4 维修：量化计算（略）

### 在NMR模块中

1. 如果能及时发现并替换损坏部件（时间0），可以带来多大的收益？
2. 如果维修需要时间T，如何评估收益？
3. 对于具体的系统设计，如何发现错误？

## 13.5 冗余技术应用：软件和数据

# 多版本软件容错

## 安装3个软件副本能表决容错吗？

- 满足故障独立性吗？

## 多个独立团队开发不同版本？

- 多版本编程（N-version programming）
- 实验结果不理想，故障仍不独立
  - 为什么？

# 数据状态分离容错

## 观察

1. 错误是不可避免的
2. 软件状态存储于非易失存储和易失存储，出错时维修是不现实的
3. 运行的程序状态分为两类
  - 系统失效时可抛弃的
  - 系统失效时需要保持的（完整性相关）

## 方法：

- 确保易失存储是可以随时抛弃的，非易失存储是受到保护的

# 存储持久性的衡量

**持久性 (Durability) 指标：操作完成后结果保持的时间长度**

1. 不长于线程的生存期
  - 例：工作目录，寄存器、cache
2. 不长于非易失存储介质的使用寿命
  - 例：RPC记录nonce，编辑器的临时文档
3. 相当于非易失存储介质的使用寿命
  - 例：文件、数据库
  - 持久性存储 (durable storage)，TTF不稳定，常实施独立多副本。
4. 数倍于非易失存储介质的使用寿命
  - 例：档案管理

## 案例：磁盘容错设计

**低成本、大容量、非易失**

**分层设计**

- 原始存储：硬件
- 快速失败：硬件+固件 = 磁盘控制器
- 谨慎存储：磁盘控制器中的固件
- 持久存储：高可用存储系统

**技术变迁：**

- 硬盘原生接口+驱动程序容错 → 容错大部分移入固件=硬驱

# 磁盘有哪些错误？

- 1. 高精度、紧公差，随时间出现老化
- 2. 尘粒等，引起磨损
- 3. 撞击可引起磁头碰撞，引起较大范围损坏
- 4. 电子元件老化，导致数据读出不稳定或读不出
- 5. 磁道臂元件损坏，导致寻道错误
- 6. 写入中途发生断电，导致写入内容只有部分更新
- 7. 写入中途操作系统崩溃，导致写入错误的内容



受干扰，写入/读取错	S
盘面坏，写入错	H
数据写入后损坏，读取错	S
寻道错误	
断电，只写入了一部分	
崩溃，写入了错误的数据	

# 磁盘容错设计

## ① 原始磁盘存储器

- 功能：
  - 1. 寻道：RAW\_SEEK (track) // Move read/write head into position.
  - 2. 写入：RAW\_PUT (data) // Write entire track.
  - 3. 读出：RAW\_GET (data) // Read entire track.
- 容错：无

受干扰，写入/读取错	×		
盘面坏，写入错	×		
数据写入后损坏，读取错	×		
寻道错误	×		
断电，只写入了一部分	×		
崩溃，写入了错误的数据	×		

# 磁盘容错设计

## ② 快速失败磁盘存储器

- 磁盘控制器中的硬件和固件，将磁道划分为扇区
- 按扇区进行**检错**
- 功能
  - $status \leftarrow FAIL\_FAST\_SEEK(track)$
  - $status \leftarrow FAIL\_FAST\_PUT(data, sector\_number)$ 
    - 写后验证
  - $status \leftarrow FAIL\_FAST\_GET(data, sector\_number)$ 
    - 出错返回出错码

## ② 快速失败磁盘存储器

### 容错

- 已检测错误
  1. 读数据后，检错码校验不通过
  2. 写数据后，检错码校验不通过
  3. 寻道读数据后，磁道号不符合
  4. 绕过校验写数据，读取校验不通过
  5. 写入时断电，未来读取校验不通过
- 未容忍错误
  1. 写入时OS崩溃
  2. 数据损坏后，恰好可以通过校验（概率忽略不计）

受干扰，写入/读取错	x	!	
盘面坏，写入错	x	!	
数据写入后损坏，读取错	x	!	
寻道错误	x	!	
断电，只写入了一部分	x	!	
崩溃，写入了错误的数据	x	x	

# 磁盘容错设计

## ③ 谨慎磁盘存储器 (Careful Disk Storage)

- 在磁盘控制器中，通过重试等方式屏蔽错误

- 功能

▸ status ← CAREFUL\_SEEK (track)

▸ status ← CAREFUL\_PUT (data, sector\_number)

▸ status ← CAREFUL\_GET (data, sector\_number)

```
procedure CAREFUL_GET (data, sector_number)
  for i from 1 to NTRIES do
    if FAIL_FAST_GET (data, sector_number) = OK then
      return OK
  return BAD
```

```
procedure CAREFUL_PUT (data, sector_number)
  for i from 1 to NTRIES do
    if FAIL_FAST_PUT (data, sector_number) = OK then
      return OK
  return BAD
```

## ③ 谨慎磁盘存储器

### 容错

- 已容忍错误

▸ 读写软错误、寻道错误

- 已检测错误

▸ 硬错误

▸ 写入时断电，未来读取校验不通过

- 未容忍错误

▸ 写入时OS崩溃

▸ 数据损坏后，恰好可以通过校验（概率忽略不计）

受干扰，写入/读取错	×	!	✓
盘面坏，写入错	×	!	
数据写入后损坏，读取错	×	!	
寻道错误	×	!	✓
断电，只写入了一部分	×	!	
崩溃，写入了错误的数据	×		

# 磁盘容错设计

## ④ 持久性存储：RAID 1

- 已检测的硬错误(永久损坏)，能否屏蔽？
  - RAID 1，思路：
    - 多副本：分别存于不同磁盘
    - 间接层：使用虚拟扇区号，RAID控制器做映射

### 功能

- status ← DURABLE\_PUT (data, virtual\_sector\_number)
- status ← DURABLE\_GET (data, virtual\_sector\_number)

## ④ 持久性存储：RAID 1

### 容错

- 已容忍错误
  - 谨慎存储层报告的硬错误
- 未容忍错误
  - 所有副本都损坏
  - 写入时OS崩溃
  - 数据损坏后，恰好可以通过校验（概率忽略不计）
  - 写入时断电？（如果多盘依次操作，可恢复一致）

受干扰，写入/读取错	×	!	✓	✓
盘面坏，写入错	×	!	✓	✓
数据写入后损坏，读取错	×	!	!	✓
寻道错误	×	!	✓	✓
断电，只写入了一部分	×	!	!	✗
崩溃，写入了错误的	×	×	×	×



# 如何检测系统崩溃导致的错误？

**目前还剩下内核崩溃的错误未能容忍，为什么不能？**

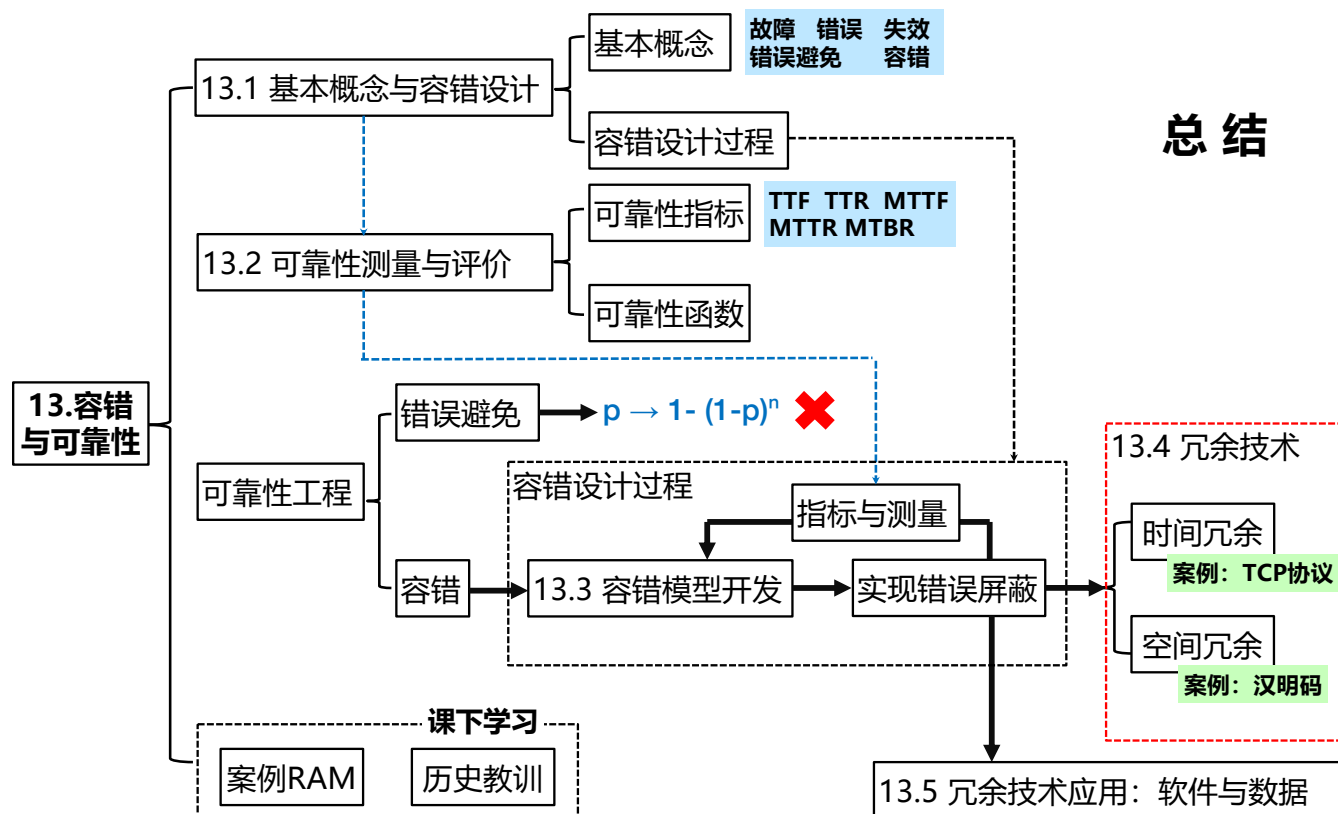
- 数据的损坏发生在数据传输给硬盘之前
- 只能从内核或应用入手解决
  - 恢复（第14章：原子性与隔离）

## 仍然有更多的威胁

**已考虑的硬盘的单个损坏，但仍有问题：**

- 例如
  - 如何让PUT成为原子操作？（第14章：原子性与隔离）
  - 如果发生灾害？（异地）
  - 如何保证异地一致性？（第15章：分布式原子性与一致性）

**更大大的威胁：技术正确，但是你用对了没有？**



## 更多

以上内容仅仅是容错入门，还有很多没有涉及的：

- 例如：
  - 恶意行为 (Byzantine faults)
  - 实用技术
  - 更复杂的需求

### 深入学习与讨论

- 阅读参考文献
- 第14-16章，将讨论如何恢复正确状态、对抗恶意行为等机制。

# 案例与历史教训

## 课下自学（已发Bb平台）

- 应用：CMOS RAM的容错模型
- 历史教训

## 习题

- 下课后发放



## 本章相关的参考文献



- Gray J N, Siewiorek D P. High-availability computer systems[J]. Computer, 1991, 24(9): 39–48.
- Siewiorek D P. Architecture of fault-tolerant computers[J]. Computer, 1984, 17(8): 9–18.
- Engler D, et al. Bugs as deviant behavior: A general approach to inferring errors in systems code[C]. Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, Operating Systems Review, 2001, 35(5): 57–72.
- Swift M M, et al. Recovering device drivers[C]. Proceedings of the Sixth Symposium on Operating Systems Design and Implementation, 2004: 1–16.
- Schroeder B, Gibson G A. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?[C]. Proceedings of the Fifth USENIX Conference on File and Storage Technologies, 2007: 1–16.
- Pinheiro E, Weber W D, Barroso L A. Failure trends in a large disk drive population[C]. Proceedings of the Fifth USENIX Conference on File and Storage Technologies, 2007: 17–28.

# 第13章 结束

