EEL-4736/5737
Principles of Computer System Design

Lecture Slides 18

Textbook Chapter 8

Fault Tolerance in Systems Design

Introduction

- Modularity:
  - Importance also from the standpoint of controlling propagation of errors
  - With proper modularity, design principles can be applied to build reliable systems from unreliable components
- We will overview fault-tolerance concepts and techniques that apply to the design of fault-tolerant systems

Core steps

- Error detection
  - First step is to be able to discover that there is an error in a value
  - Apply *redundancy* (e.g. replicas, coding)
- Error containment
  - Limit the propagation of an error
  - Careful application of modularity
- Error masking
  - Ensure correct operation despite error
  - Additional redundancy – to discover what is the correct value from the erroneous value: error correction

Examples

- Enforced modularity
  - Client/server, virtual memory, …
  - Primary purpose – error containment
- Network data link layer
  - Error detection through checksums
- End-to-end protocols
  - Timeout, resend – mask loss of packet
- Real-time applications may fill-in missing data through interpolation
  - Masking data loss

Terminology and basics

- Terms used: faults, failures, errors
  - Distinction involves modularity
- Fault
  - Underlying defect, imperfection, flaw that has potential to cause problems
  - Example: software fault
    - Programmer types < where algorithm requires <= in a variable comparison
    - Fault may not cause incorrect behavior depending on input combinations
      - But may cause system to crash – a failure

Faults

- Hardware fault
  - Stuck-at-zero gate
  - Until a module depends on the gate producing 1, nothing goes wrong
- Design fault
  - Wrong calculation leading to design of a buffer smaller than needed to accommodate requests according to specification
  - Fault may not be lead to problems until there is congestion
- Environment fault
  - Lightning strike; radiation, alpha particles

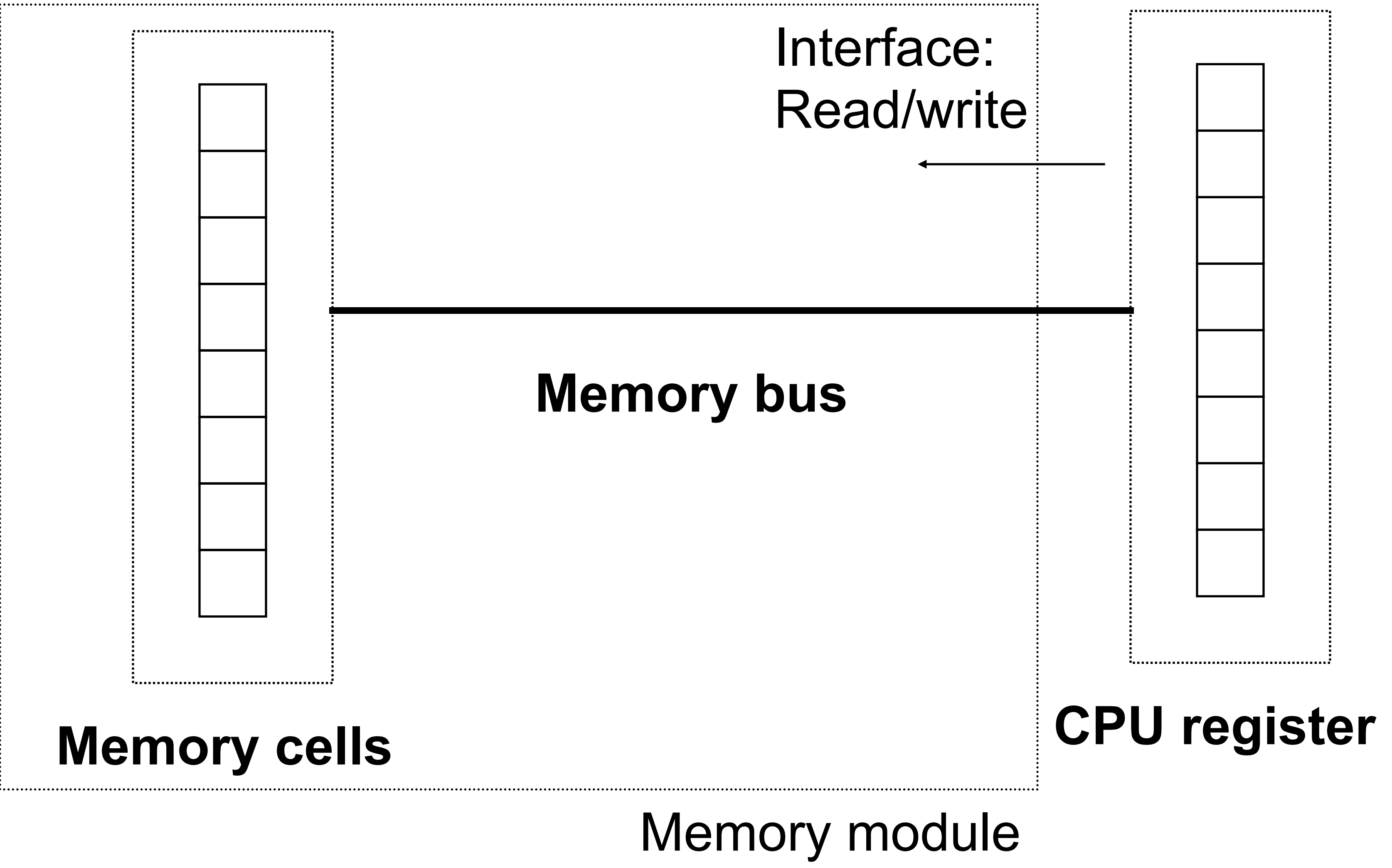
Faults and errors

- *Latent* fault:
  - Not affecting correct behavior
- *Active* fault:
  - Wrong values appear in control signals, data values
  - These wrong results are called *errors*
- Example:
  - A memory cell with a bit flipped due to an alpha particle

Failures

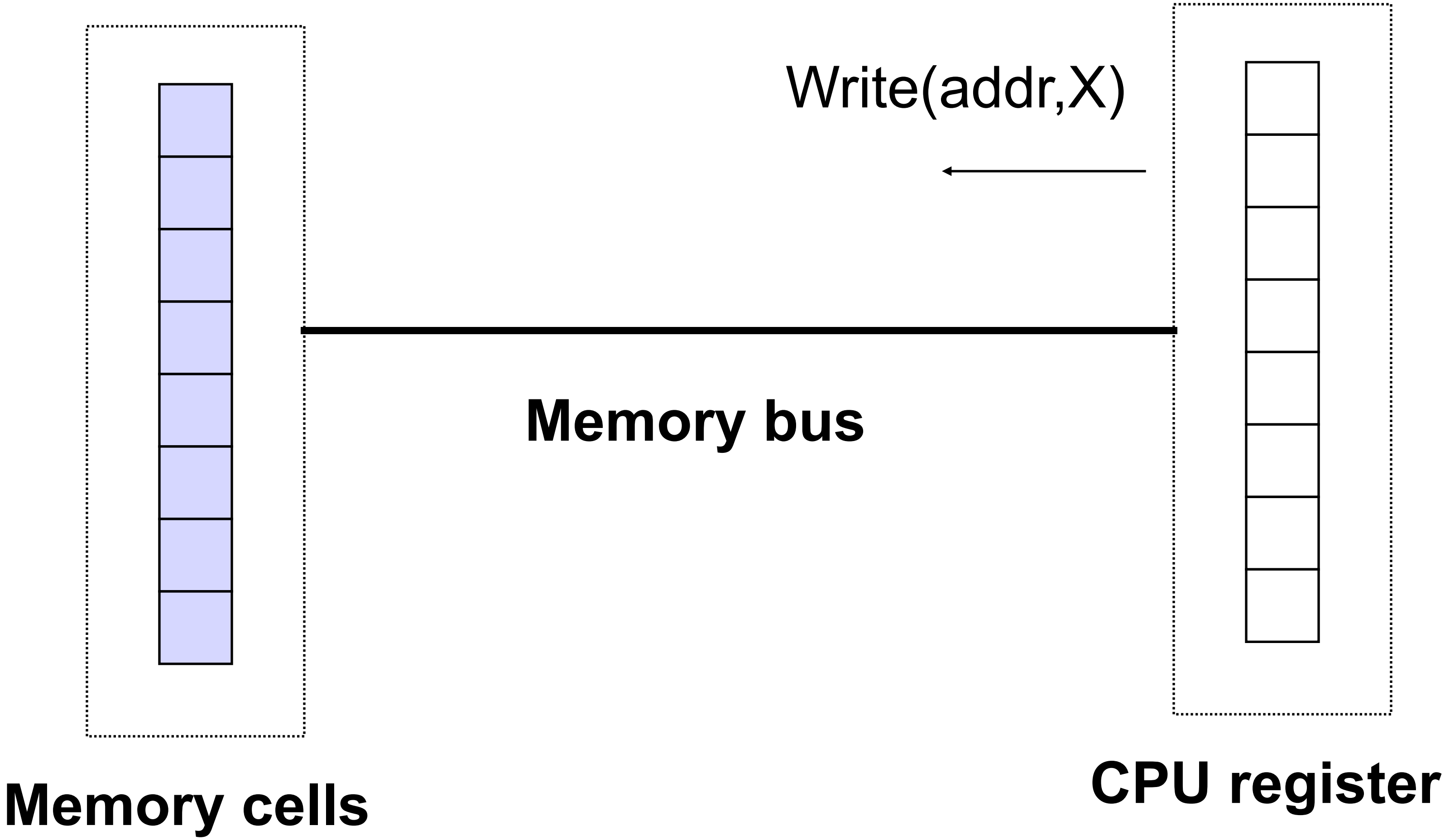
- If an error is not detected and masked, can lead to a module not performing to specification
- *Failure* – not producing the intended result at an interface
- Distinction between fault and failure is closely tied to modularity
  - Failure of a subsystem is a fault from the point of view of the subsystem that contains it
  - Subsystem fault may cause error that leads to failure of the larger subsystem
  - Larger subsystem may be designed to anticipate the possibility of faults, detect and mask errors

Example

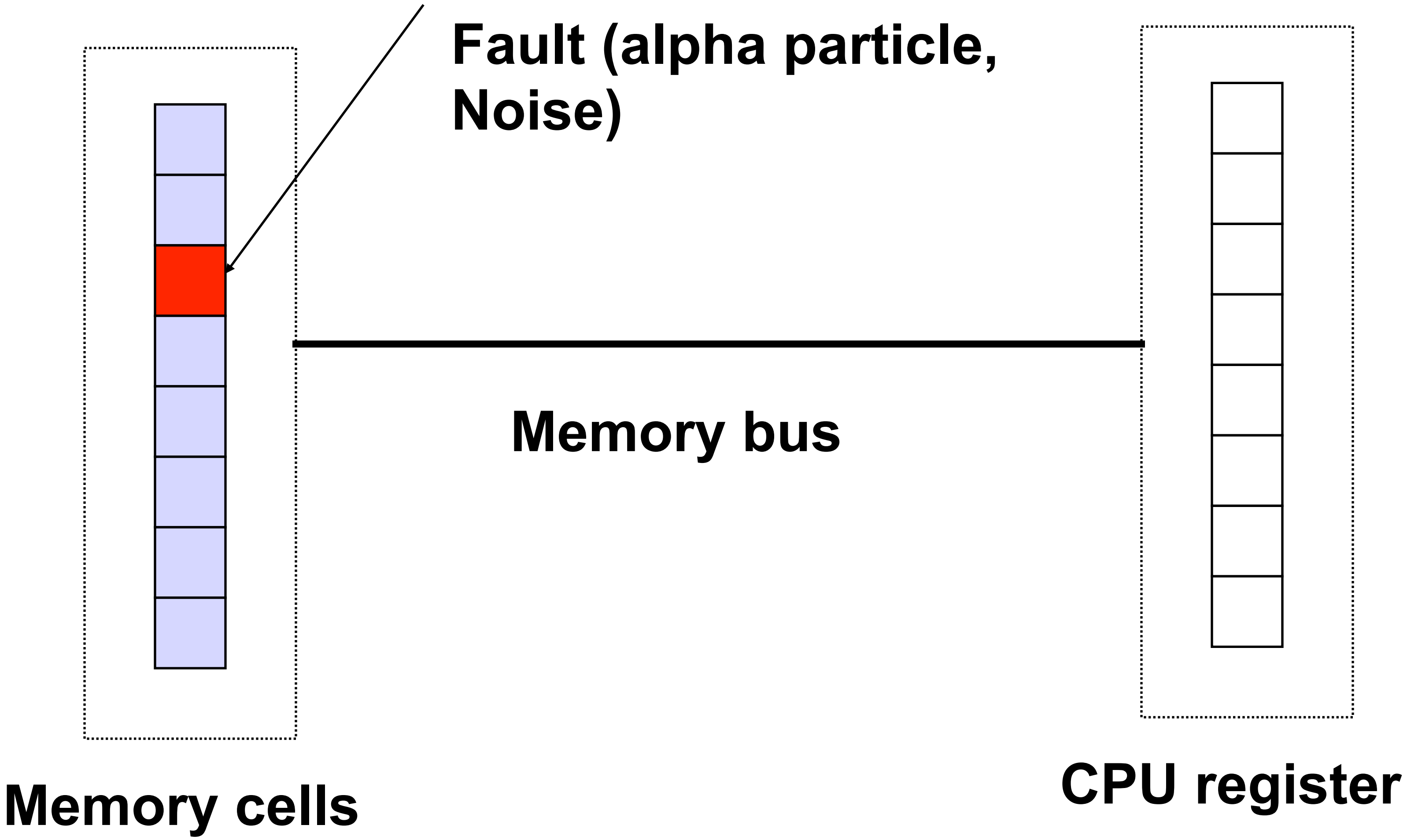


Interface:
Read/write
Memory bus
Memory cells
CPU register
Memory module

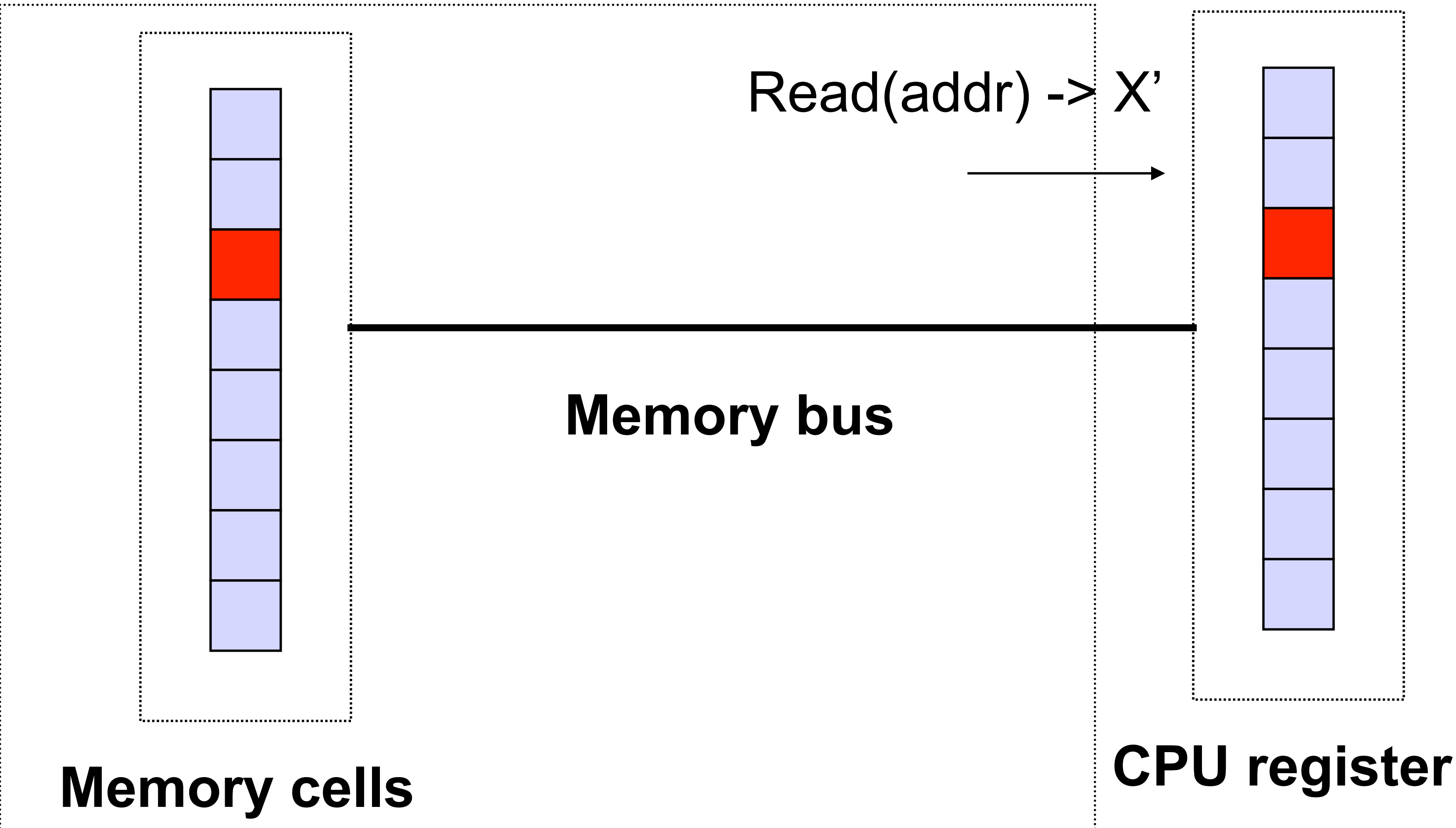
Example



Example


Fault (alpha particle, Noise)
Memory bus
Memory cells
CPU register

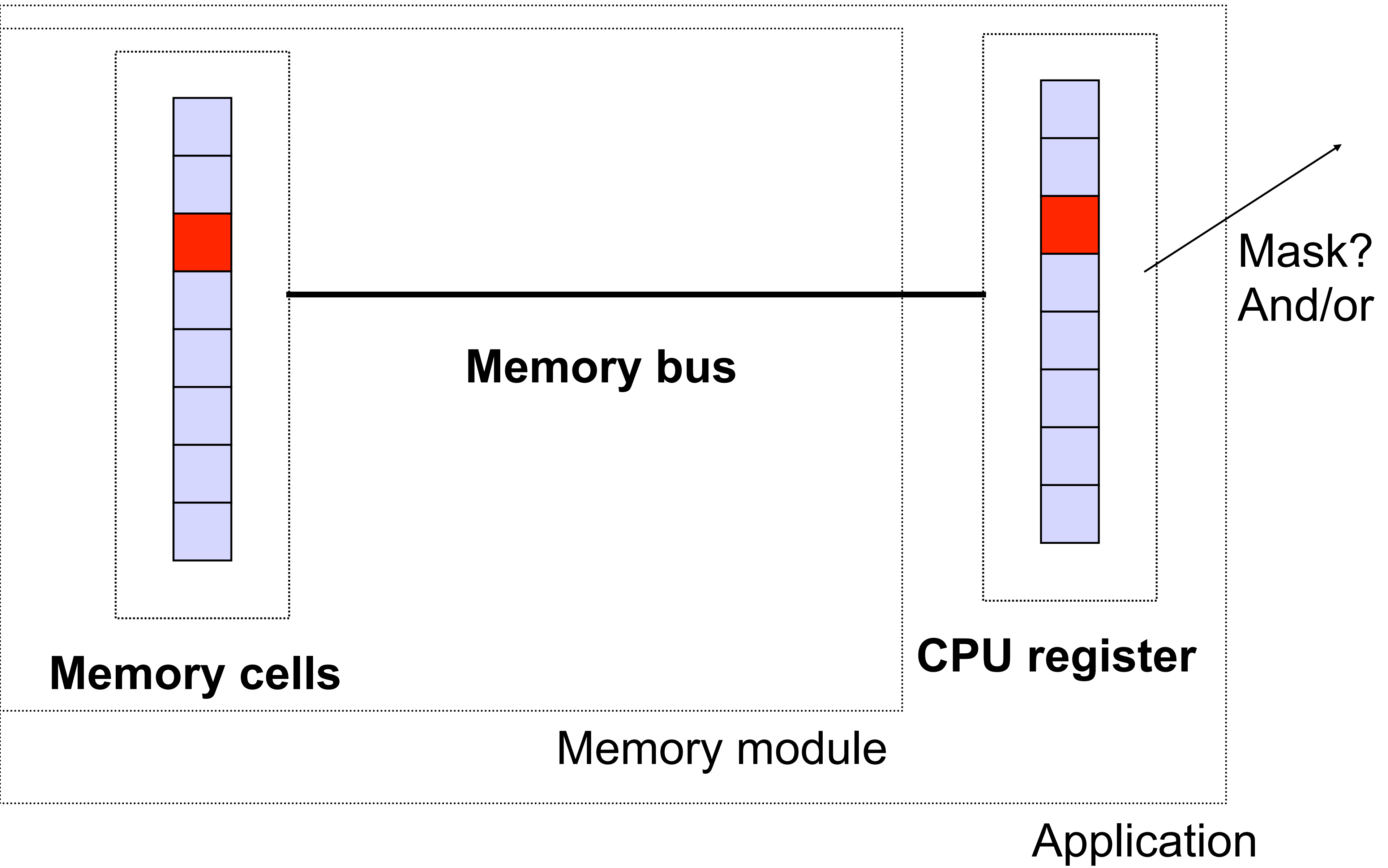
Example



Memory module

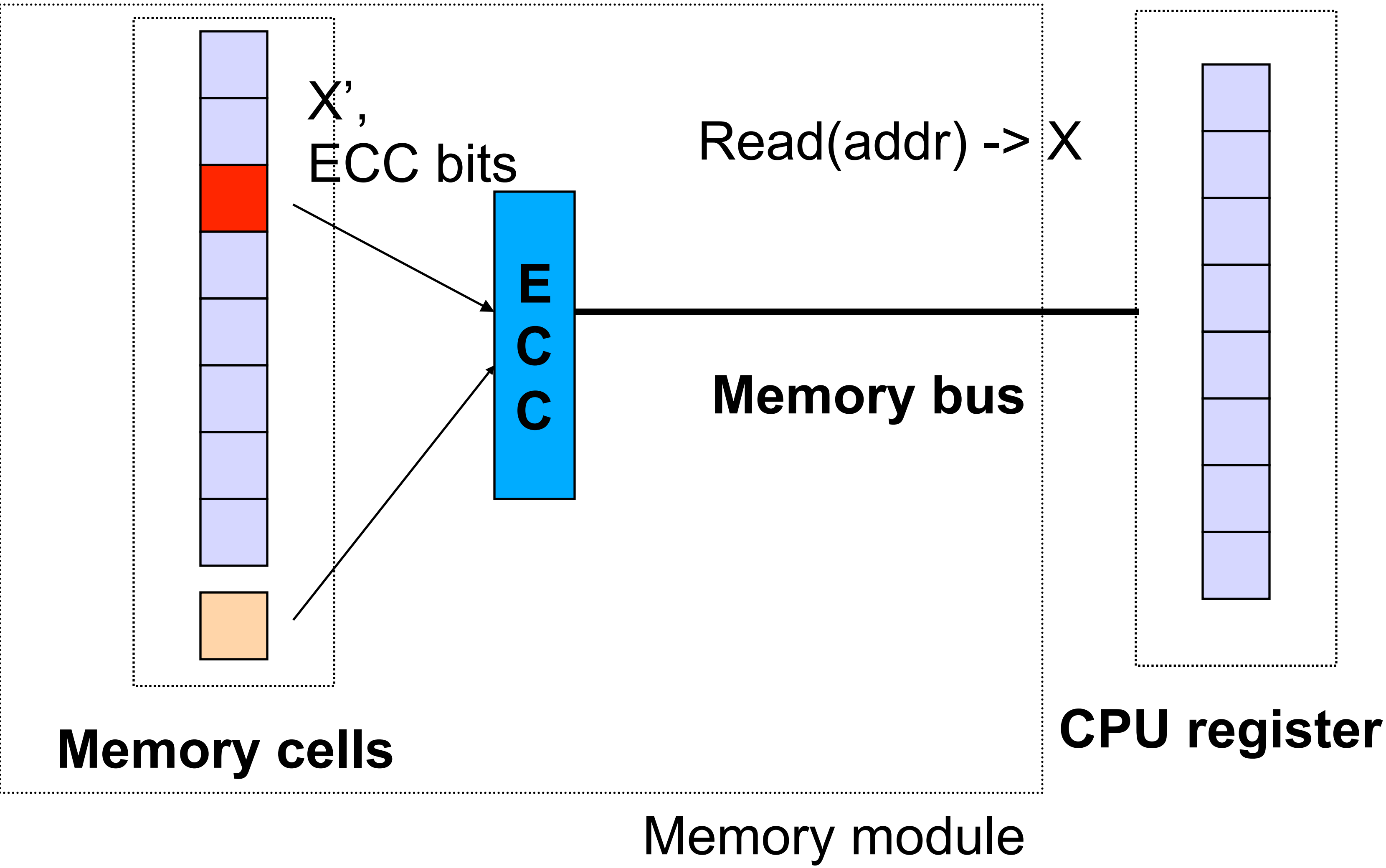
Failure in the memory module

Example



Mask?
And/or
Memory bus
Memory cells
CPU register
Memory module
Application

Example



Error masked in memory module

Fault tolerance

- Detecting active faults and component subsystem failures and performing an action in response
  - Correct an error
  - Contain an error
- Failures appear at the interface of a subsystem
  - Boundary adopted for error containment is usually boundary of the smallest subsystem where it occurred

Containing errors

- From the perspective of a higher-level subsystem H, one of its lower-level subsystems L may contain an error in different ways:
  - L masks error – H does not perceive anything went wrong
  - L detects error and reports at interface – *fail-fast* design
  - L immediately stops, limiting propagation of bad values – *fail-stop*
    - H needs to take additional measures, e.g. use a timeout to detect error in L
    - Problem: can be difficult to distinguish failure from slow progress
  - L may do nothing – e.g. continue running and creating wrong values at its interface

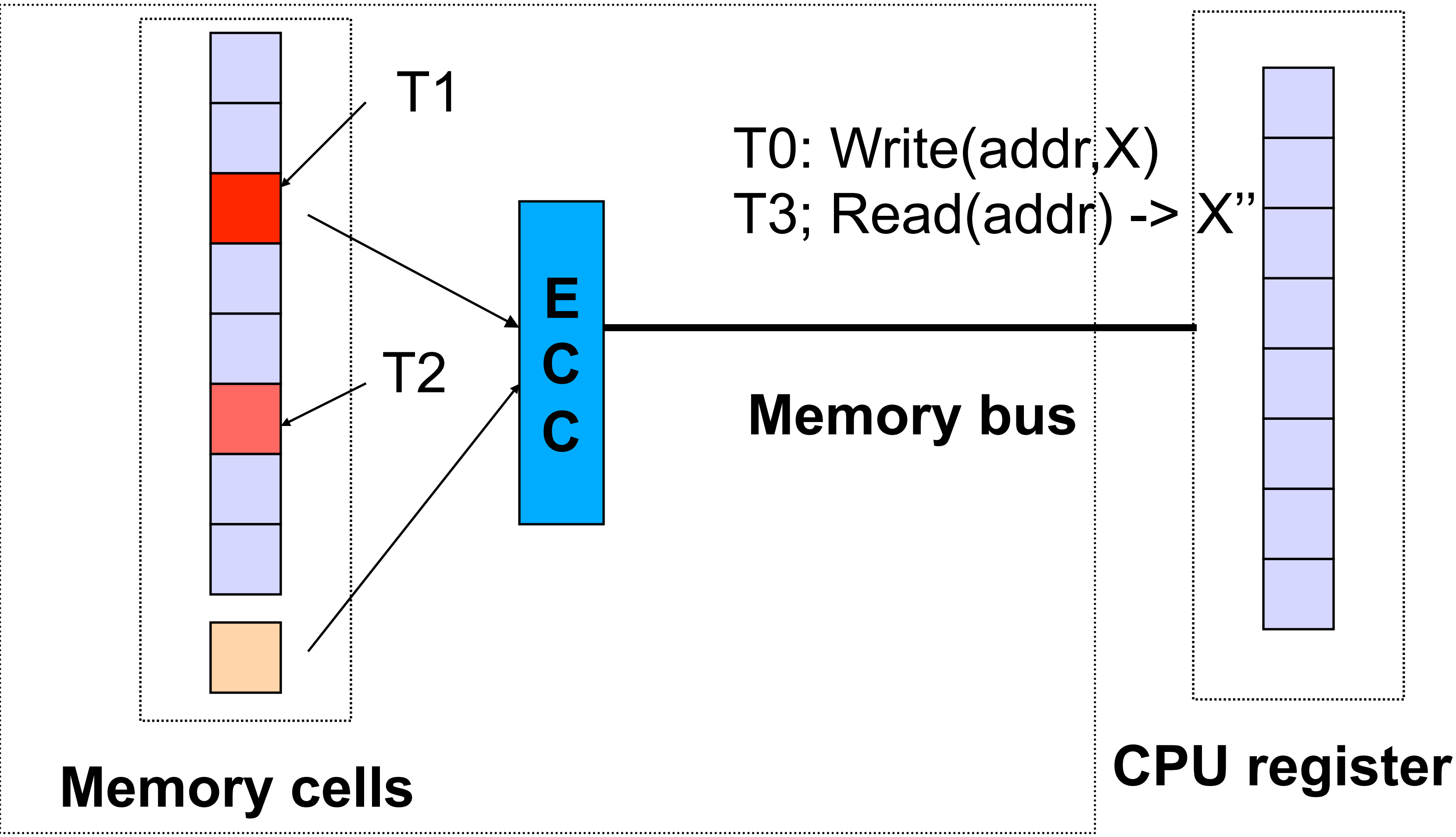
Fault types

- Another distinction that is important
  - Faults may be *persistent* or *transient*
  - E.g. stuck-at-zero gate vs. alpha particle
- Transient fault also referred to as single-event upset (SEU) – temporary occurrence
- A transient error that can be successfully masked by retry – *soft error*
- Persistent fault continues to produce errors no matter how many retries – *hard error*
  - Intermittent fault – persistent fault but active only occasionally (e.g. at high noise levels)

Detection latency

- Time between a fault causing an error, and the error being detected or causing module to fail
  – Important because error masking mechanisms may depend on having a single or small number of errors at a time
  – “At a time” depends on detection latency
  – If latency is high, may create opportunity for additional errors to accumulate

Example



Memory module

Error would have been masked if T3 < T2

Example

- Improperly fabricated memory cell transistor stuck at zero: persistent fault
  - Whenever the bit should contain 1, the fault is active, and the value is in error
  - Whenever the bit should contain 0, the fault is latent
- If chip is part of memory module with error correction codes, error will not propagate
  - Single-event bit-flip upsets will not propagate either – unless errors line up to exceed number that can be detected and corrected

Fault-tolerant design process

- One approach – fault avoidance
  - Design a reliable system entirely with components that are so reliable that their chance of failure is within specification
  - Cost constraints; not generally possible in large systems
    - Probability of failure of system with N components, each with failure probability p

      $P_{system} = 1-(1-p)^N$
- Alternative: fault-tolerant approaches

Fault-tolerant design process

- Develop a fault model
  - Identify every potential fault
  - Estimate the risk of each fault
  - When risk is high, design methods to detect resulting errors
- Apply modularity to contain damage from the high-risk errors
- Design procedures that can mask detected errors
  - Temporal redundancy: retry operations on same component
  - Spatial redundancy: use multiple components for the same operation

Fault-tolerant design process

- Update the fault-tolerance model to account for improvements
- Iterate until probability of untolerated faults is acceptably low
- Deploy and observe the system in the field
  - Include extensive logging in the system to observe how many errors the system is successfully masking
  - Perform “post-mortem” analysis on failures to identify all reasons for each failure
- Revise fault tolerance model, iterate

Trade-offs

- Whether probability of failures is unacceptably high or sufficiently low depends on application
  – Desktop at home, or system unattended in a space mission?
  – Business decision weighing risk versus cost; insurance

Summarizing design principles

- Be explicit on all assumptions
  - So it is know which ones are being addressed and understand limitations if assumptions change over time
- Design for iteration
  - Probabilities of failure can change dramatically over time
- Apply safety margins
  - Monitor the system to evaluate how often errors are masked
- Careful analysis of failures
  - Complex systems fail for complex reasons
- Strive for simplicity
  - Complexity increases the opportunity for errors

Measures of Reliability

- A typical scenario:
  - System works properly over a period of time; fails; failure is repaired and system restarted
  - Towards quantitative models:
    - Account for time to failure - TTF
    - Account for time to repair - TTR
    - Account for non-deterministic nature of failures

Availability

- Observe a system through N run-fail-repair cycles
  - Each cycle with TTFi, TTRi
- System availability

  Time system was running / Total time

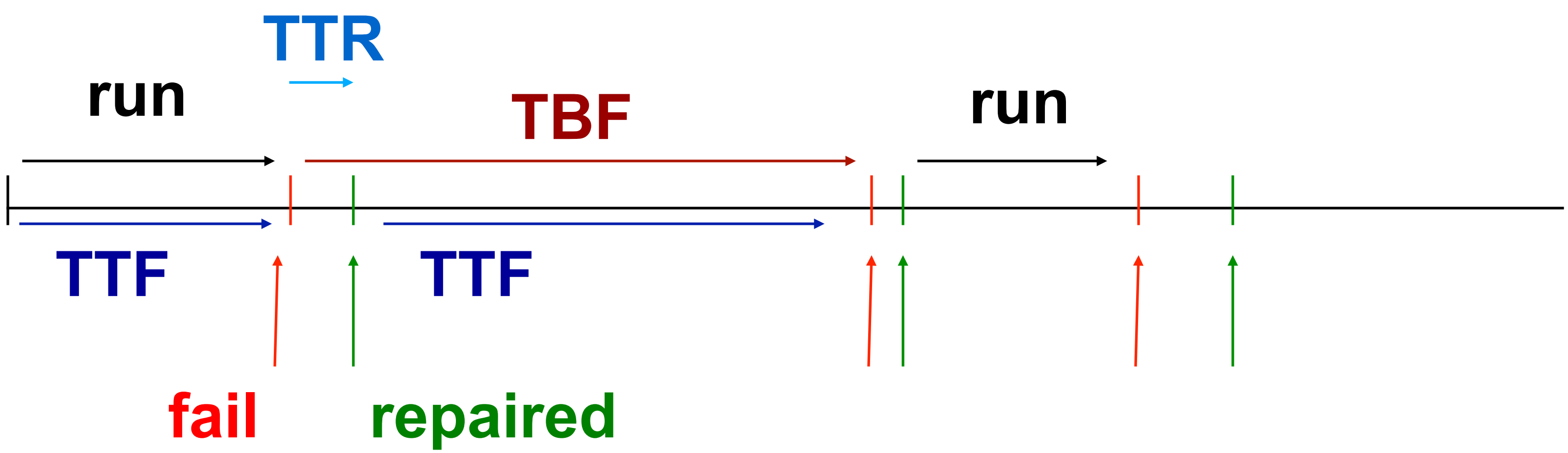
  $\Sigma_{(i=1..N)}$ (TTFi) / $\Sigma_{(i=1..N)}$ (TTFi + TTRi)
- Statistics: mean time to fail and repair:

  MTTF = 1/N $\Sigma_{(i=1..N)}$ (TTFi)

  MTTR = 1/N $\Sigma_{(i=1..N)}$ (TTRi)

Availability

- Availability = MTTF / (MTTF+MTTR)
- Mean time between failures: MTBF
  - MTBF = MTTF + MTTR
- Downtime: 1 – Availability = MTTR/MTBF



Measuring MTTF

- It would take a long time to go through multiple run-fail-replace trials to measure MTTF
- Ensemble approach:
  - Estimate average by running "N" independent run-fail trials
  - May not be accurate for a failure process; e.g. potential impact of repair on future failures is not captured

Using MTTF/MTTR measures

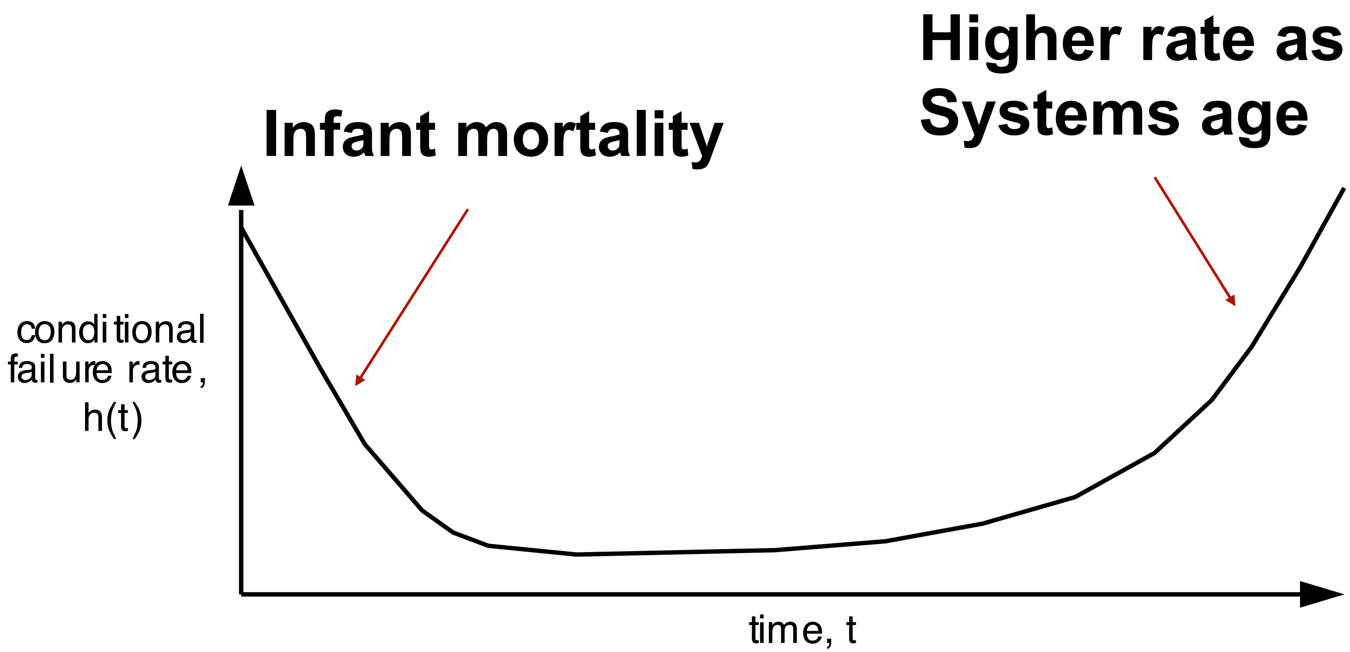
- These statistics can be potentially used:
  - To evaluate how a deployed system is doing
    - Backward-looking; measures summarize observed behavior
  - To make predictions of expected behavior
    - Forward-looking; past samples are at times good predictors of future behavior, but not always
- MTTF – takes long time to measure for systems that are already very reliable
  - Need to resort to proxy measurements and modeling

Example

- Hard disk “MTTF”
  - Typical 300,000 hours, or 34 years
    - High-end disks: 1+ million hours
  - Can’t afford to wait for either time average nor ensemble approach
  - Run a large number of disks for a shorter time, count the number of failures, use this sample
- E.g. 1000 disks, 3000 hours (4 months)
  - 10 fail – 1 failure per 300,000 hours

Discussion

- Computing “MTTF” from a sample of failures in a shorter trial with many devices will not be accurate in general
- “Bathtub curve”



Reliability functions

- Bathtub curve – *conditional* failure rate
  - Probability a module fails between t and t+dt, given that the component is still working at time t
  - One way of expressing failure characteristics of a component or system
- Reliability function:
  - R(t) = Prob (module not yet failed at time t)
    - Assuming the module was operational at time 0
- Unconditional failure rate:
  - f(t) = Prob (module fails between t and t+dt)
  - MTTF = ∫(t=0,∞)t*f(t)dt

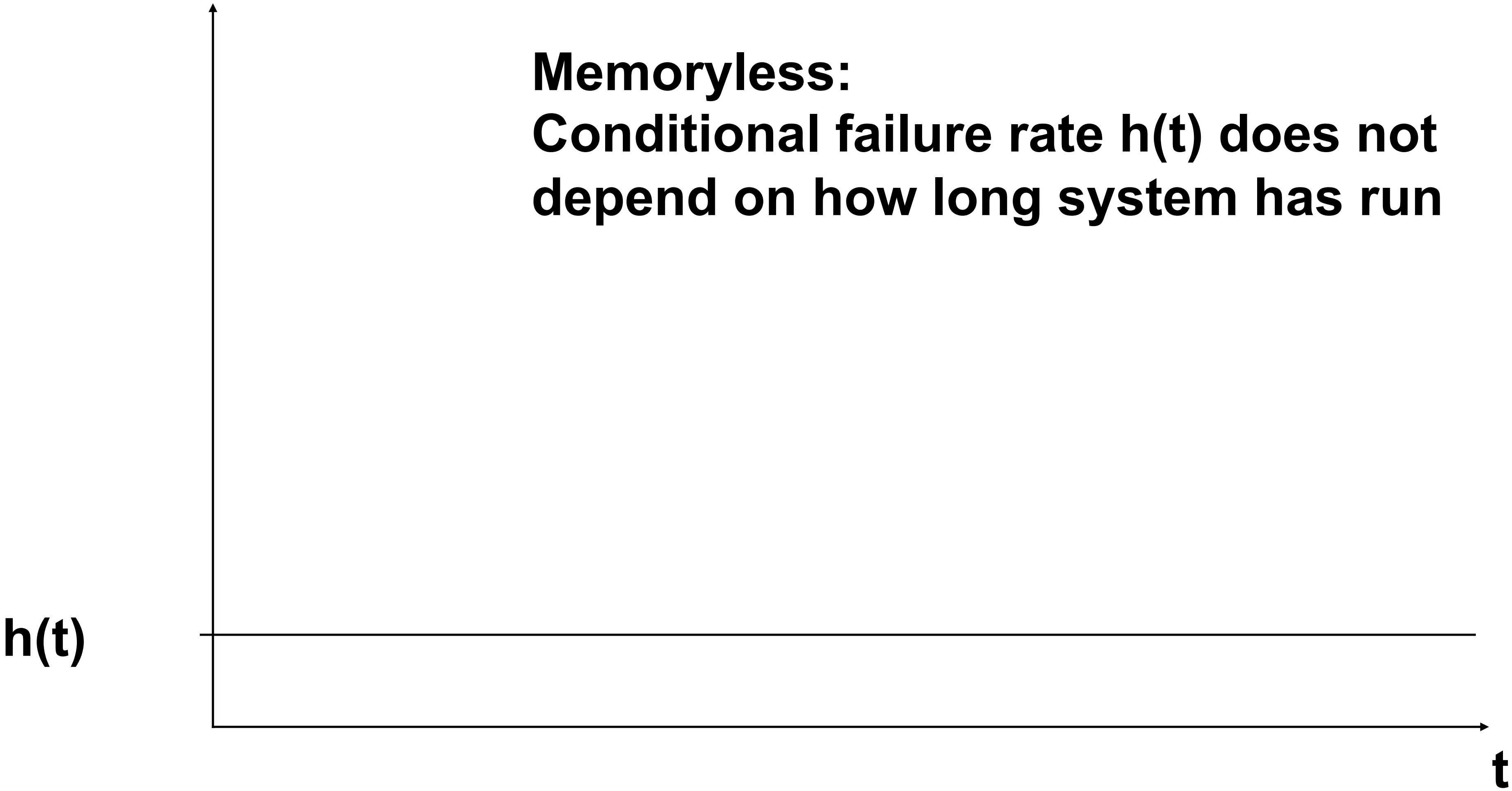
Reliability functions

- Unconditional failure rate $f(t)$ – probability density function
  - Non-negative, $\int(0,\infty)f(t)dt = 1$
- Cumulative probability function:
  - $F(t) = \int(0,t)f(t)dt$
  - Cumulative probability that a system has failed at time t
  - $R(t)$ is the cumulative probability that a system has *not* failed at time t
    - $R(t) = 1 – F(t)$
- Conditional failure rate (probability of failure between t and t + dt given it has not failed up to t):
  - $h(t) = f(t)/R(t)$

Reliability functions

- Some components/systems exhibit relatively uniform failure rates
  - Flat instead of bathtub conditional failure rate curve
  - Reliability function is an exponential; MTTF is inverse of conditional failure rate:
    - $R(t) = \exp(-t/MTTF)$
  - *Memoryless* – conditional failure rate is independent of how long the component has been operating

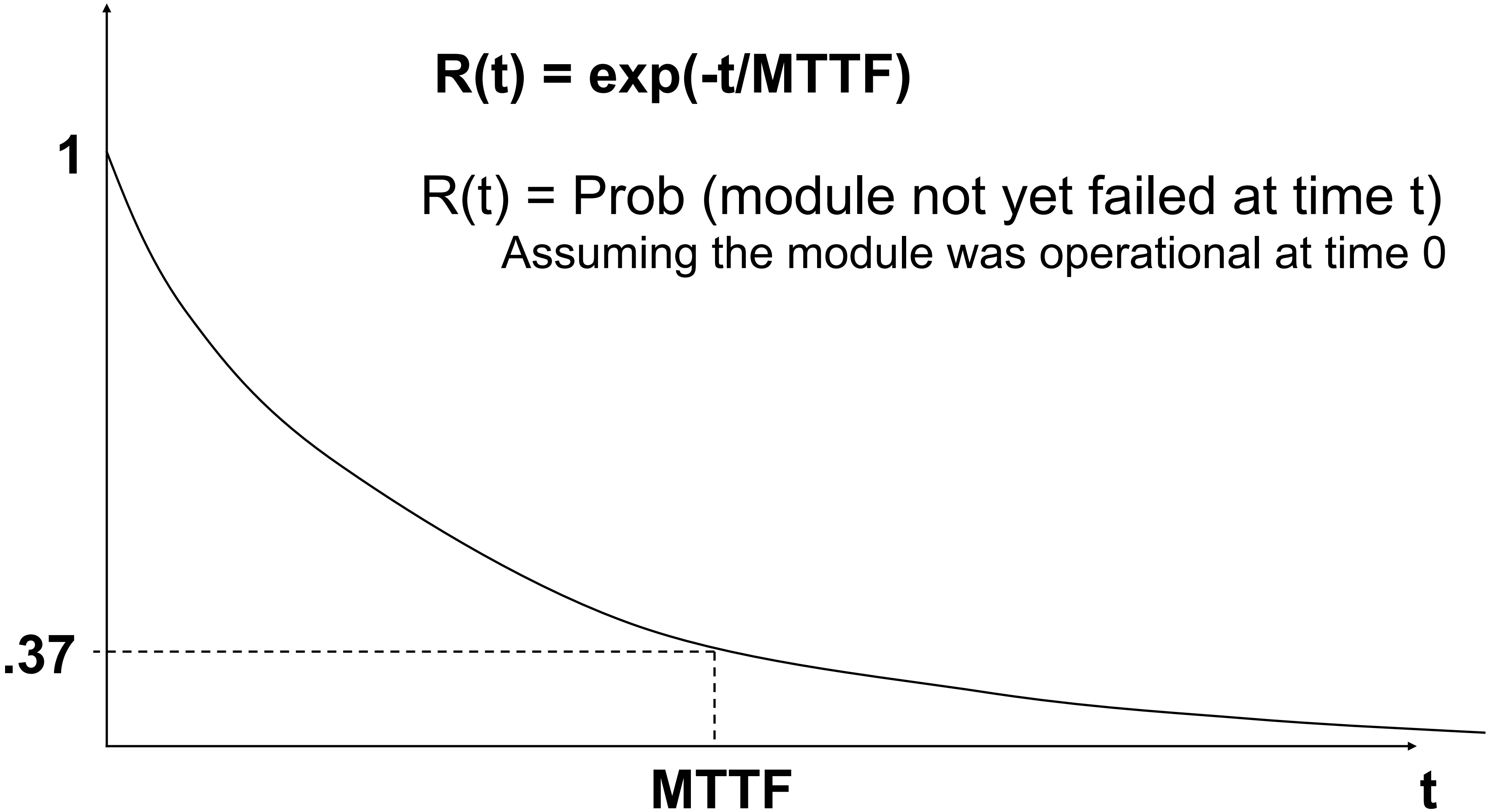
Example


Memoryless:
Conditional failure rate h(t) does not depend on how long system has run
h(t)
t

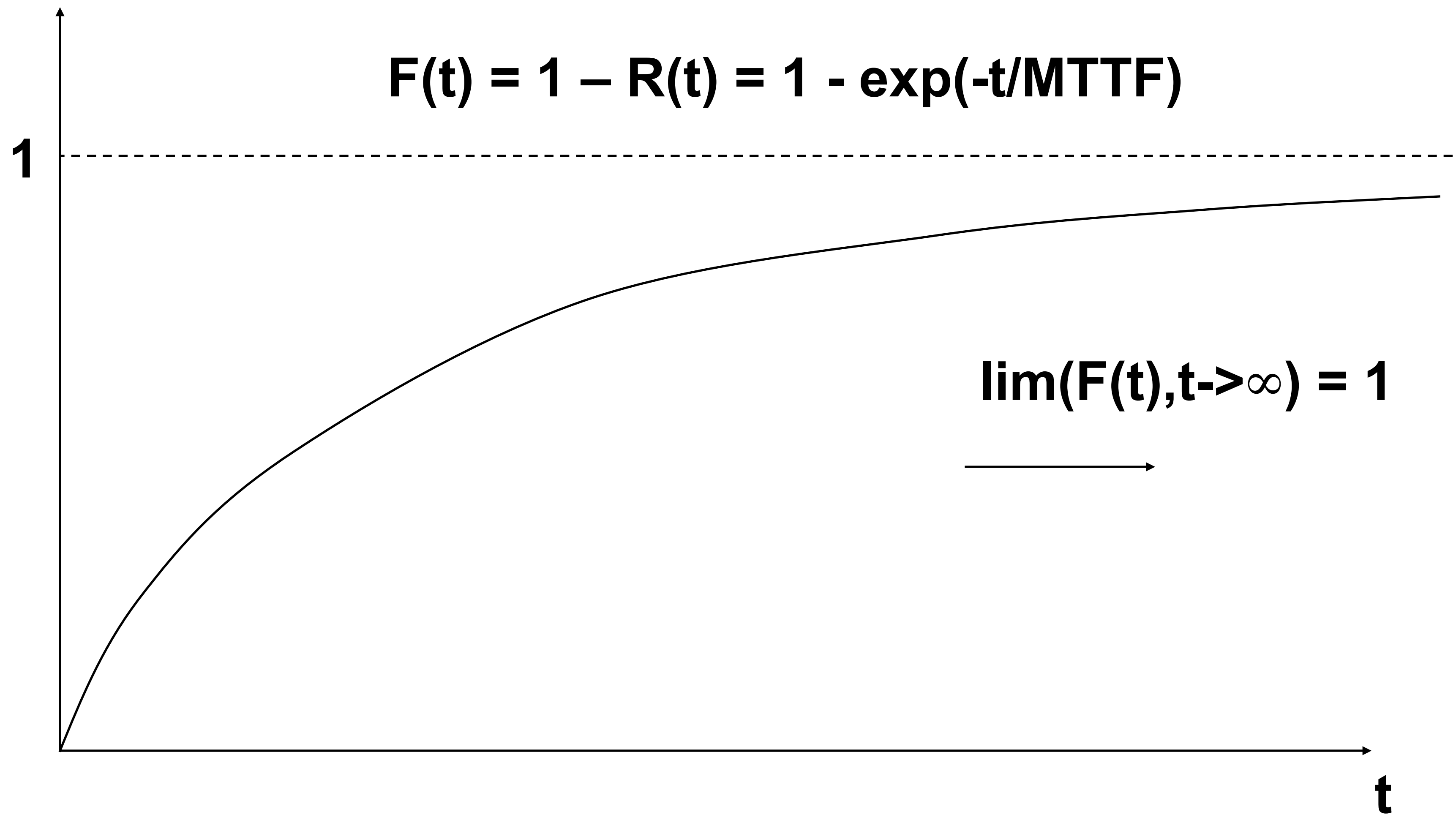
Example

R(t) = exp(-t/MTTF)

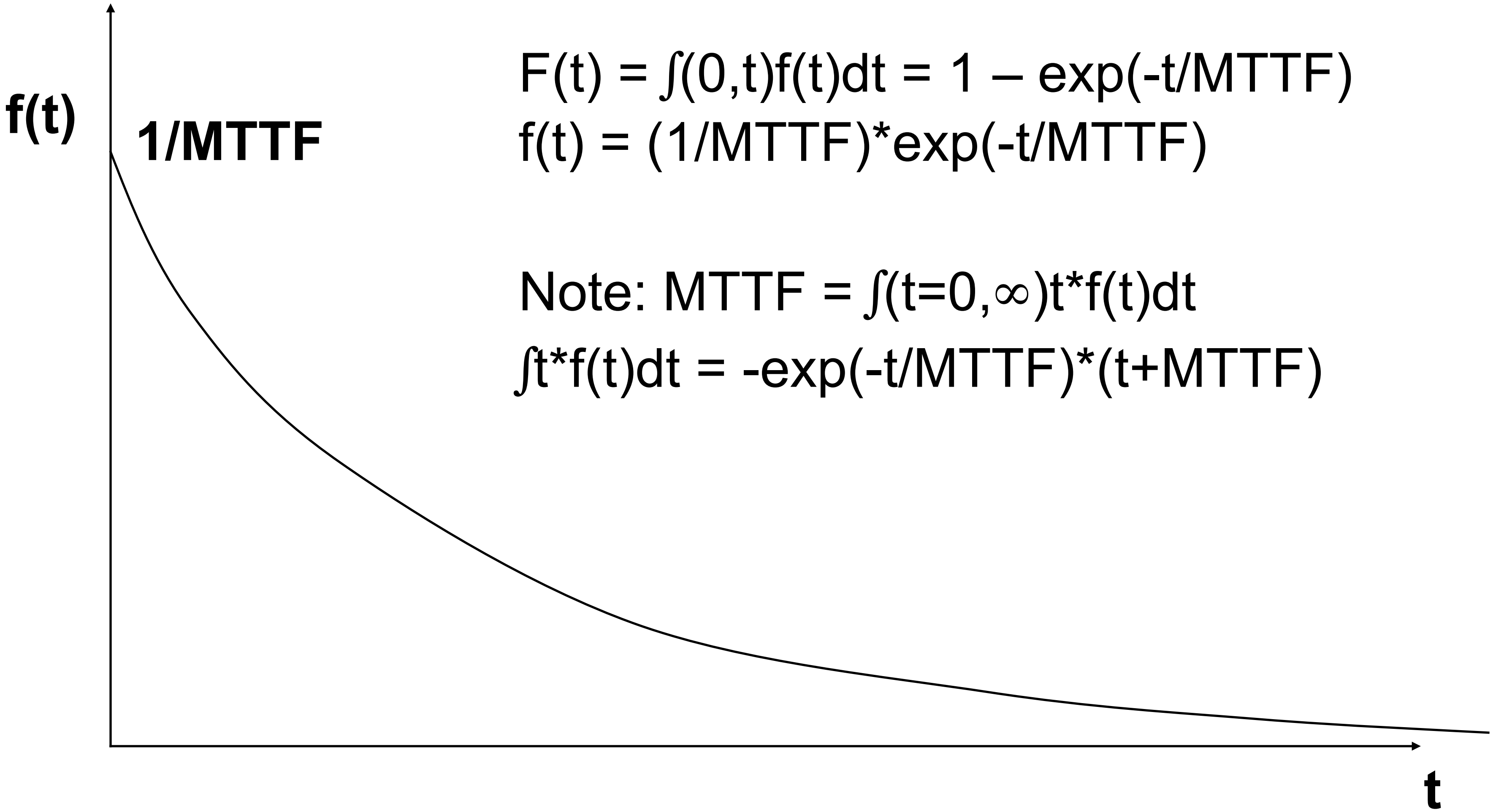
R(t) = Prob (module not yet failed at time t)
Assuming the module was operational at time 0



Example



Example



$F(t) = \int(0,t)f(t)dt = 1 - \exp(-t/MTTF)$

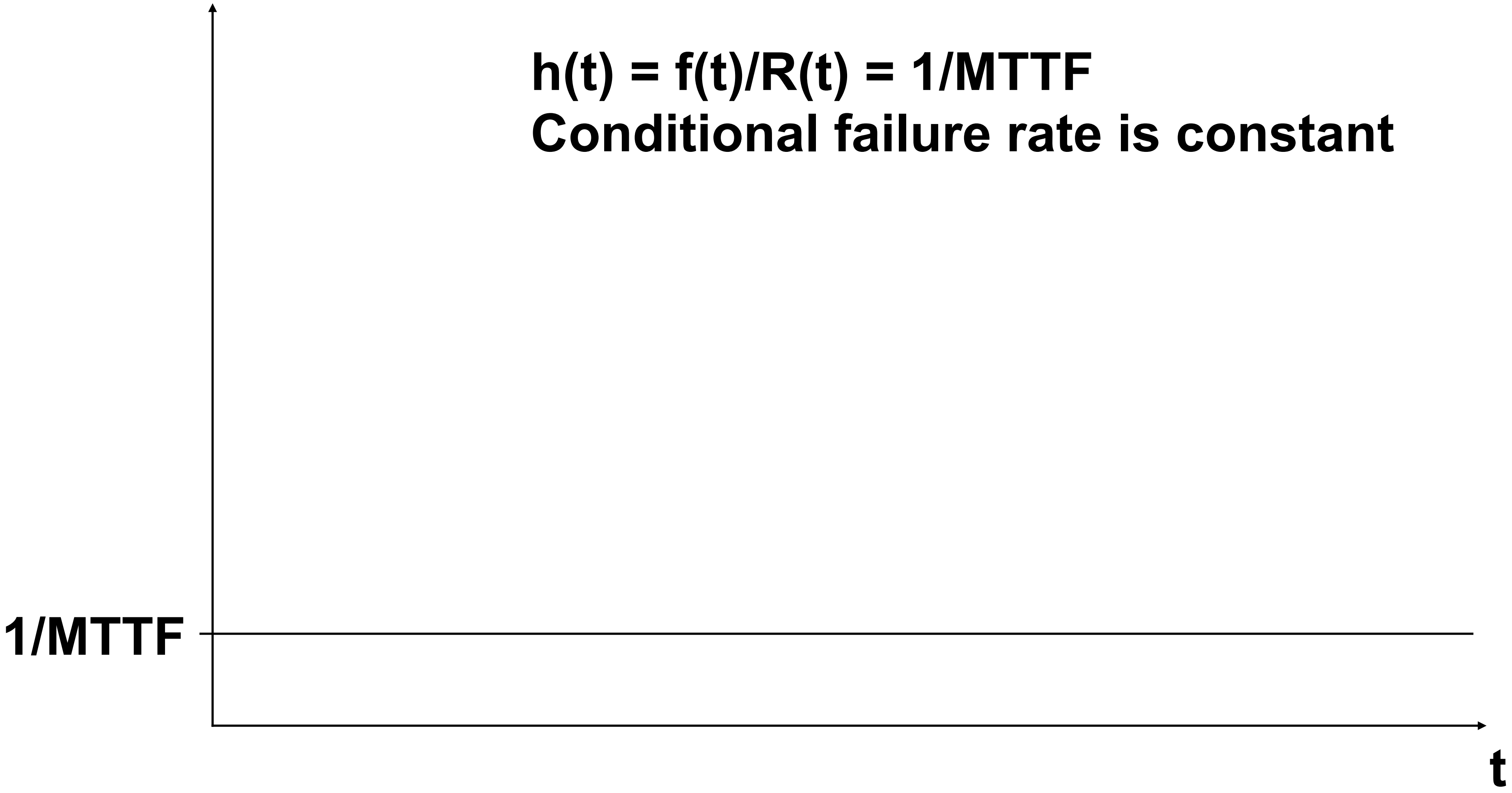
$f(t) = (1/MTTF)*\exp(-t/MTTF)$

Note: $MTTF = \int(t=0,\infty)t*f(t)dt$

$\int t*f(t)dt = -\exp(-t/MTTF)*(t+MTTF)$

Example

h(t) = f(t)/R(t) = 1/MTTF
Conditional failure rate is constant



MTTF vs. Measured Data

- Paper by Shroeder/Gibson, Usenix FAST 2007
  - Failure data collected from various clusters; over 100,000 disks. Many “HPC” clusters
  - Variance between datasheet MTTF and disk replacement rates in the field was larger than we expected. The weighted average ARR was 3.4 times larger than 0.88%, corresponding to a datasheet MTTF of 1,000,000 hours.
  - For older systems (5-8 years of age), data sheet MTTFs underestimated replacement rates by as much as a factor of 30.

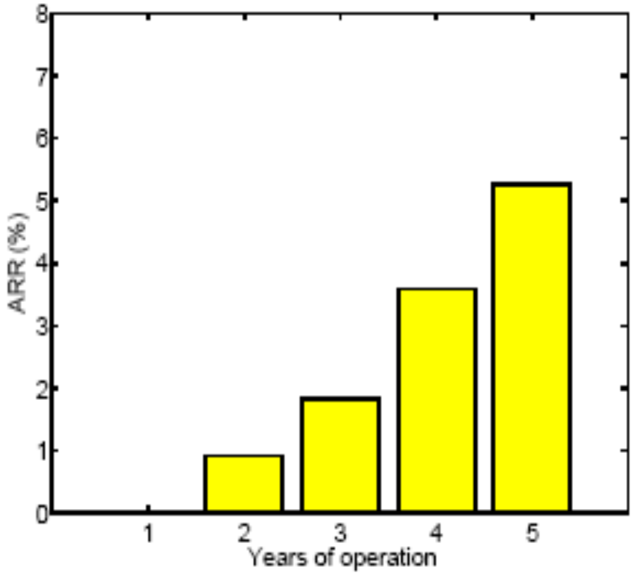
[From: Shroeder, Gibson, FAST-2007]

MTTF vs. Measured Data

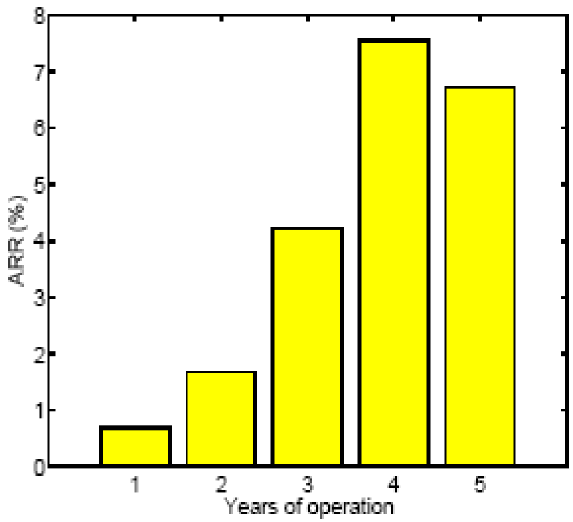
- Paper by Shroeder/Gibson, Usenix FAST 2007
  - Even during the first few years of a system's lifetime (< 3 years), when wear-out is not expected to be a significant factor, the difference between datasheet MTTF and observed time to disk replacement was as large as a factor of 6.
  - Failure rate is not constant with age; rather than a significant infant mortality effect, we see a significant early onset of wear-out degradation. That is, replacement rates in our data grew constantly with age, an effect often assumed not to set in until after a nominal lifetime of 5 years.

[From: Shroeder, Gibson, FAST-2007]

Annual Replacement Rates



HPC1 (filesystem nodes)



HPC1 (compute nodes)

[From: Shroeder, Gibson, FAST-2007]

More measured data

- Another FAST'07 paper
  - Pinheiro, Weber, Barroso; data from Google data centers
  - Failure model: "a drive is considered to have failed if it was replaced as part of a repairs procedure."
    - Since it is not always clear when exactly a drive failed, consider the time of failure to be when the drive was replaced.

[From: Pinheiro et al, FAST-2007]

Annual failure rates

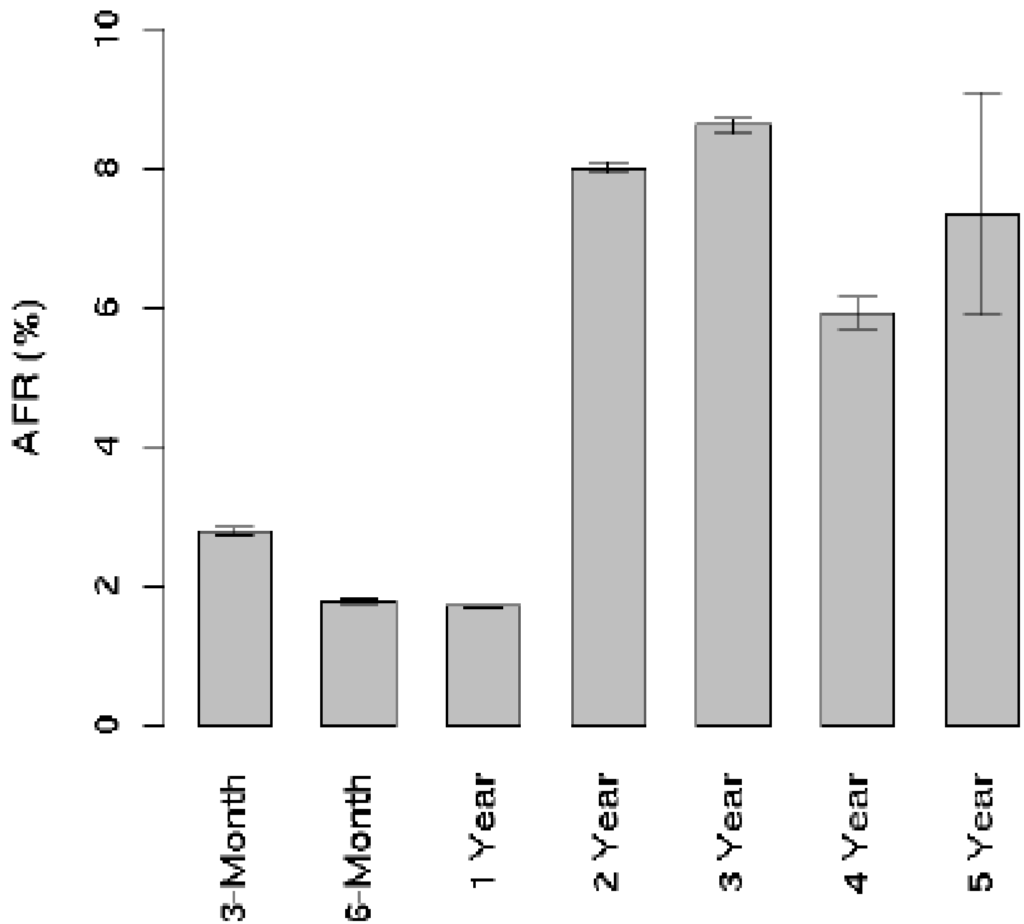


Figure 2: Annualized failure rates broken down by age groups

Baseline failure rate; average across all drives

Different drive Models

Magnitude of AFR similar to the ones observed in Schroeder/Gibson

[From: Pinheiro et al, FAST-2007]

Failure rate vs utilization

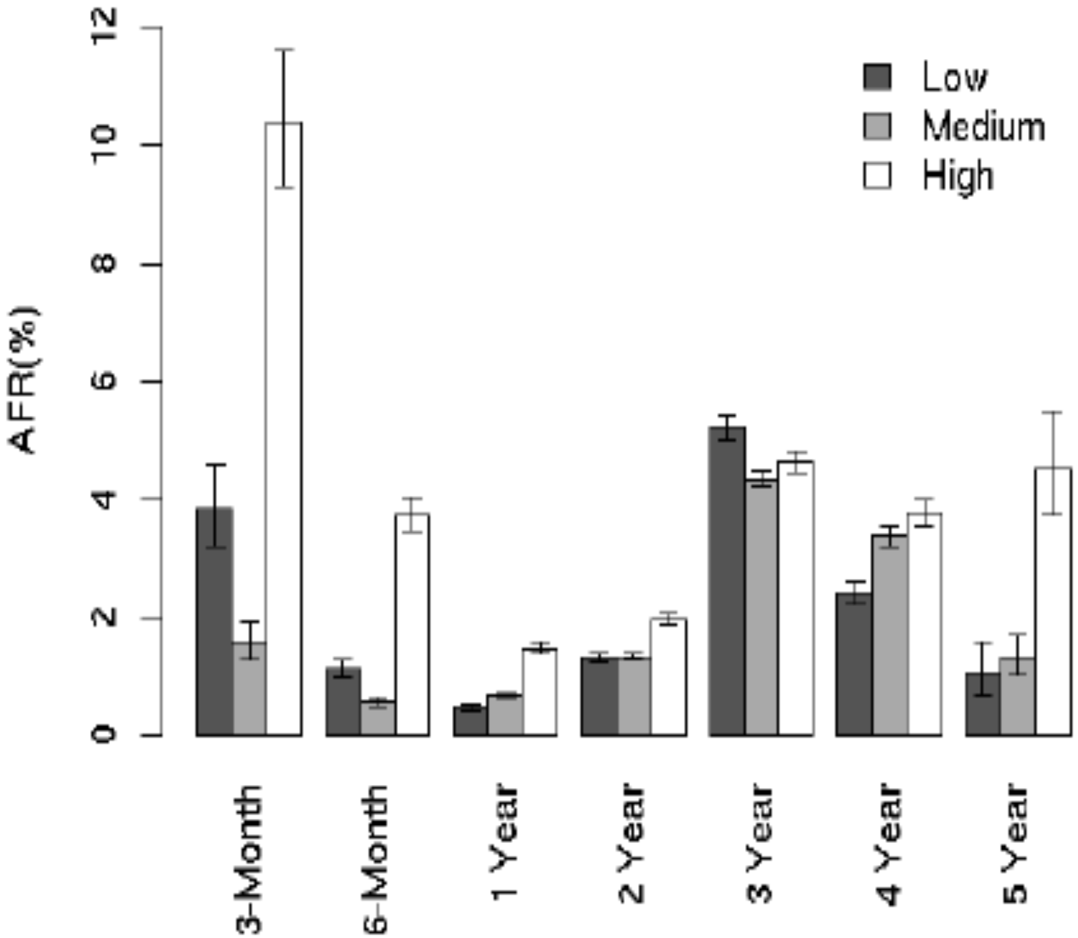


Figure 3: Utilization AFR

Weekly average of read/write Bandwidth; 25th, 50-75th, and 75th percentile

Correlation between high util and AFR noticed on 'infant' disks; those replaced do not show strong correlation later

[From: Pinheiro et al, FAST-2007]

Failure rate vs. temperature

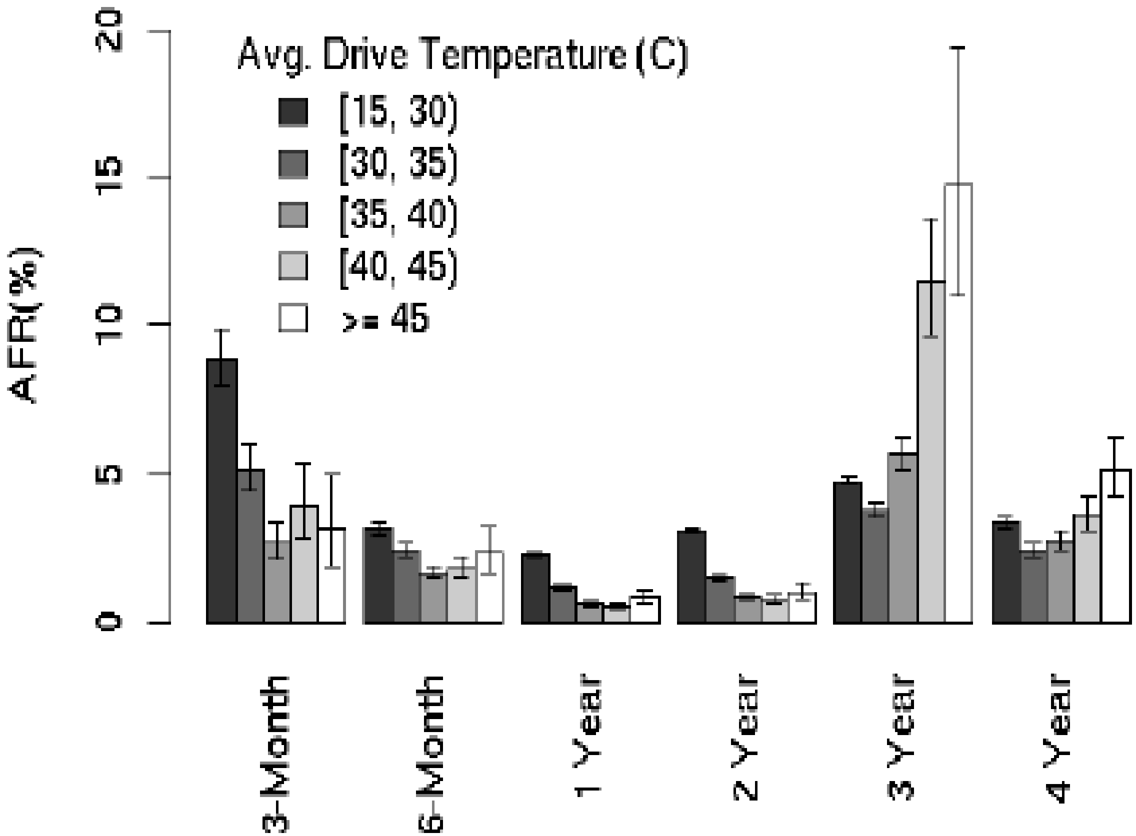


Figure 5: AFR for average drive temperature.

Early in life, lower temperature has higher correlation with AFR

Later on, trend shifts to higher temperature

High temperature an issue for older drives; at mid-life, not as important

[From: Pinheiro et al, FAST-2007]

Failure rate vs. scan errors

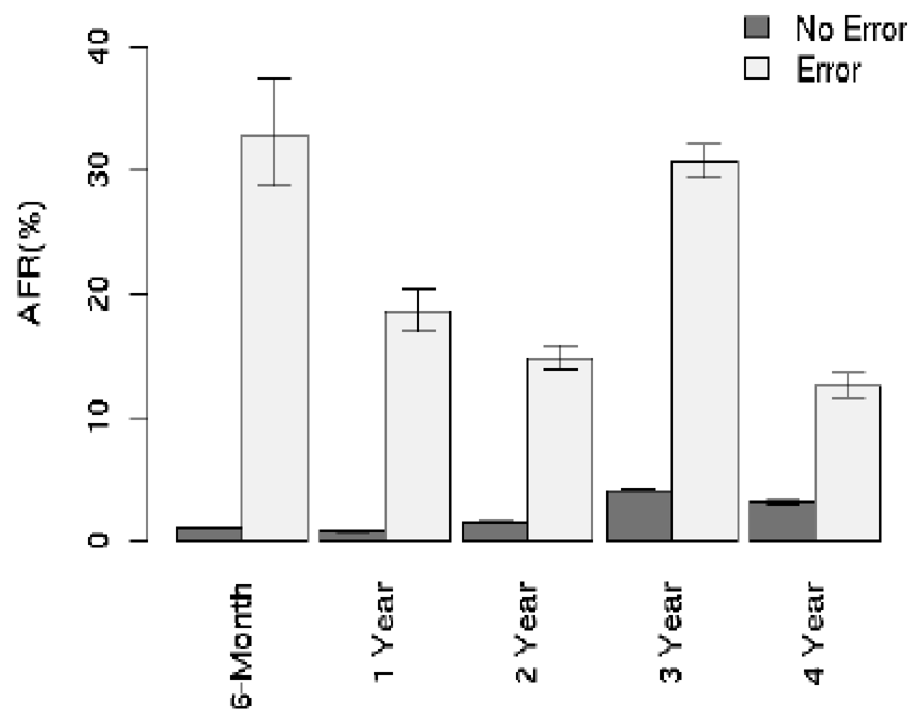


Figure 6: AFR for scan errors.

Drives perform scan checks in background and report errors as they are detected

In their studies, fewer than 2% of drives show scan errors

Drives which have reported at least one scan error have significantly higher AFR

[From: Pinheiro et al, FAST-2007]

Failure rate vs. reallocation count

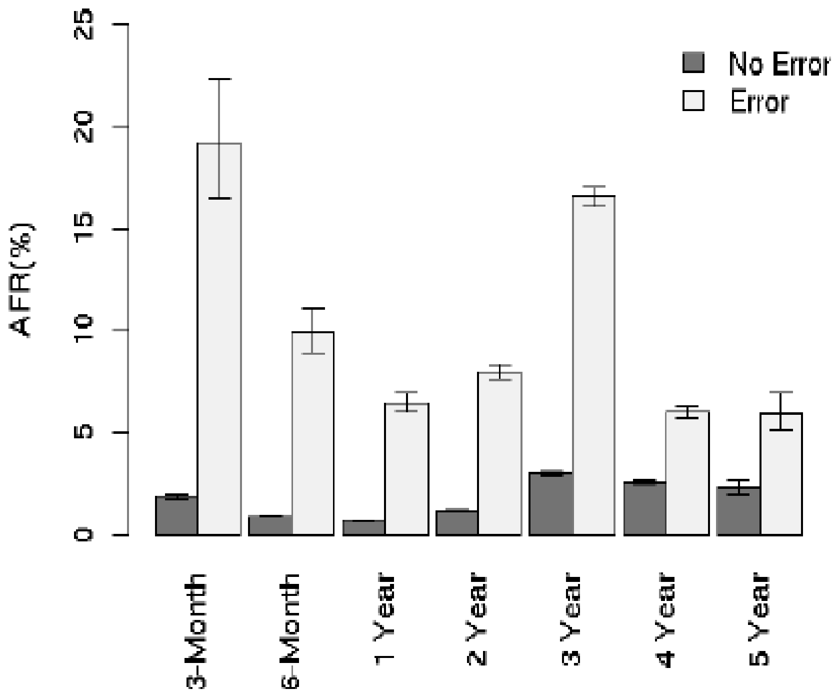


Figure 7: AFR for reallocation counts.

Drives reallocate and remap 'bad sectors' when errors are consistently detected

Presence of reallocation correlates with higher AFR

[From: Pinheiro et al, FAST-2007]

Failed population vs. errors

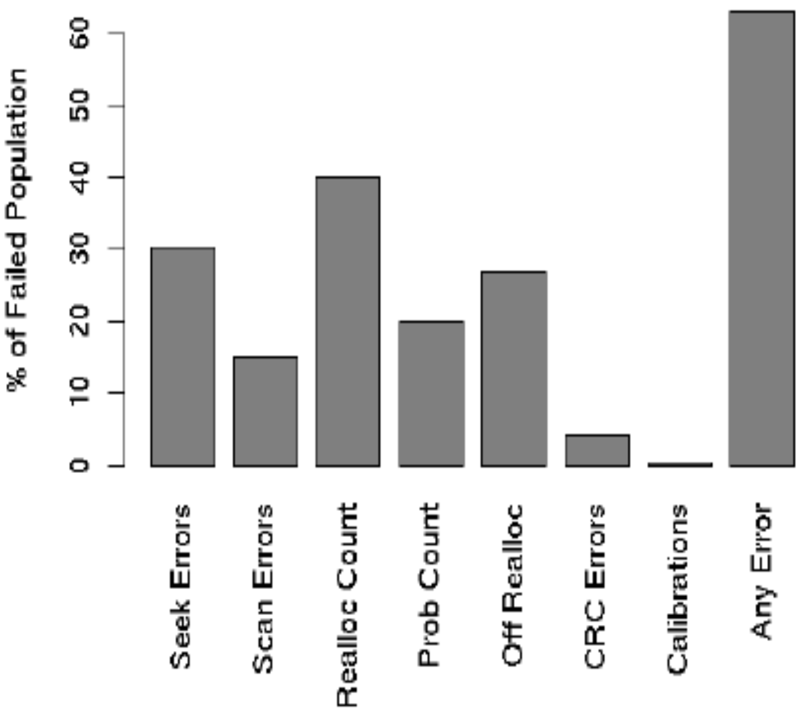


Figure 14: Percentage of failed drives with SMART errors.

Errors such as seek, scan, relocation count can be monitored for using SMART interface (self-monitoring analysis and reporting technology)

SMART reports correlate strongly with failure, but 56% of the disks fail without any of these signs – hard to predict

[From: Pinheiro et al, FAST-2007]

Responding to active faults

- Do nothing
  - Error becomes a failure of the module
  - Larger system is responsible for both discovering and handling the problem
  - In a system with several layers, errors may propagate through multiple layers before being discovered and handled
    - Containment becomes more and more difficult
  - Example: RAM without ECC

Responding to active faults

- Fail fast
  - Report at interface that something went wrong
  - Still turns the problem over to the next layer, upper layer at least has knowledge of failure
  - Example: bit-flip in RAM with parity

Responding to active faults

- Fail safe
  - Transforms incorrect values to values that are known to be acceptable, even if not correct or optimal
  - Example: failure in sequencer of intersection stop lights fail safe to blinking red lights

Responding to active faults

- Fail soft
  - System continues to operate, but with degraded subset of specifications, e.g. lower performance of missing features
  - Example: engine failure in 4-jet airplane; disk failure in RAID disk array; memory module failure reducing effective available memory of system

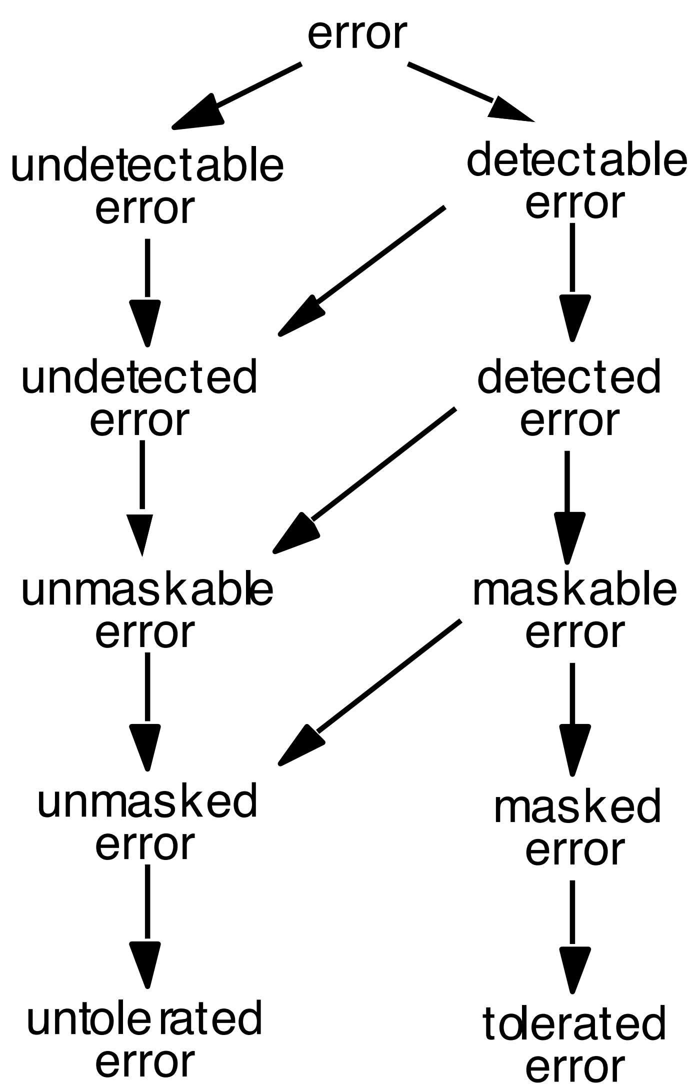
Responding to active faults

- Mask the error
  - Any value/values that are incorrect are corrected and module meets specification
  - Example: bit-flip in ECC-protected memory

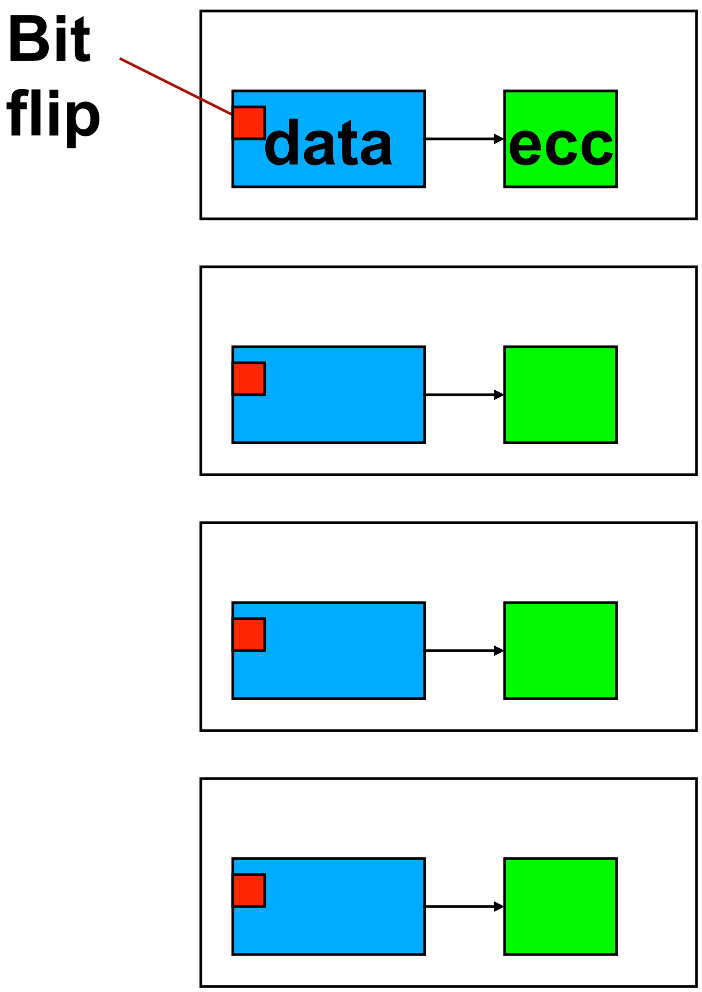
Detecting errors

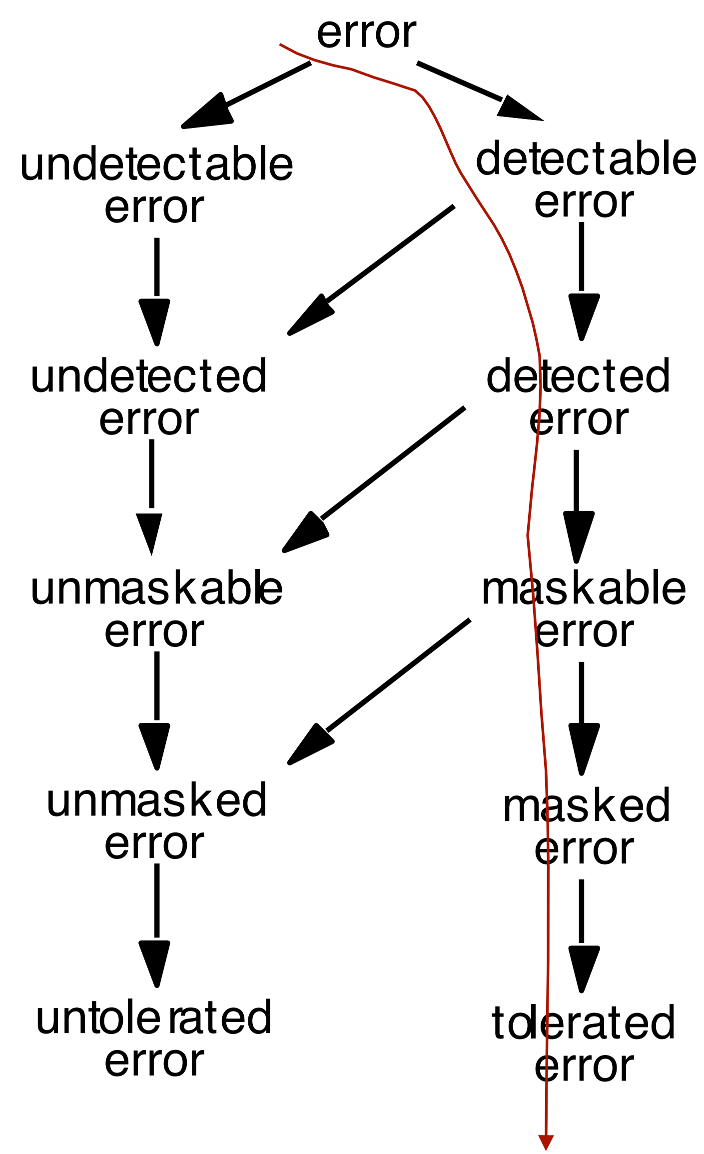
- Detecting a fault in most cases means the need to detect an error
- *Detectable* error: can be detected reliably
  - If detection procedure is in place and error occurs, it becomes a *detected* error
- *Maskable* error: one for which it is possible to devise a procedure to recover correctness
  - If masking procedure is in place and the error occurs, is detected and masked, it is a *tolerated* error
- *Untolerated* error: undetectable, undetected, unmaskable, unmasked

Tolerating errors



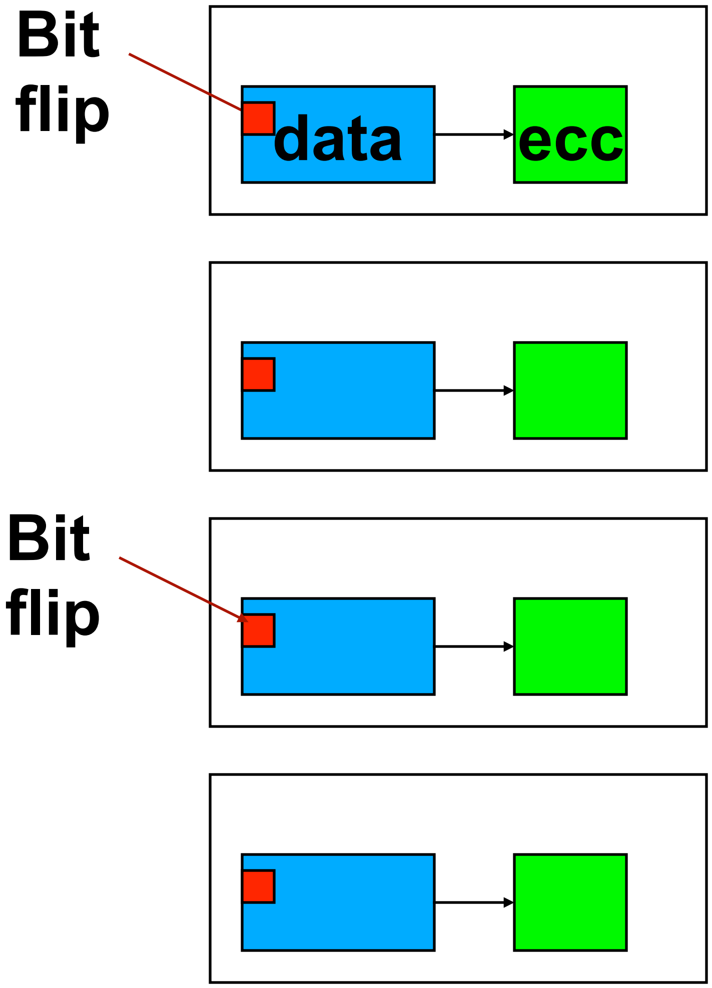
Example

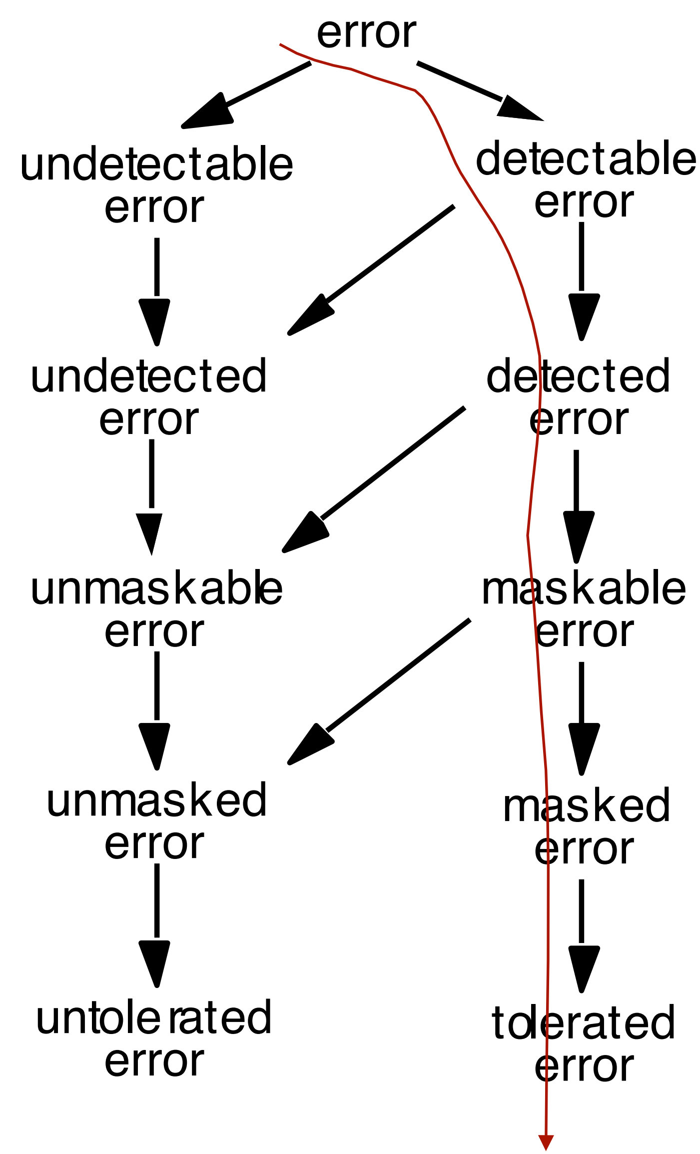




- Data needed by application

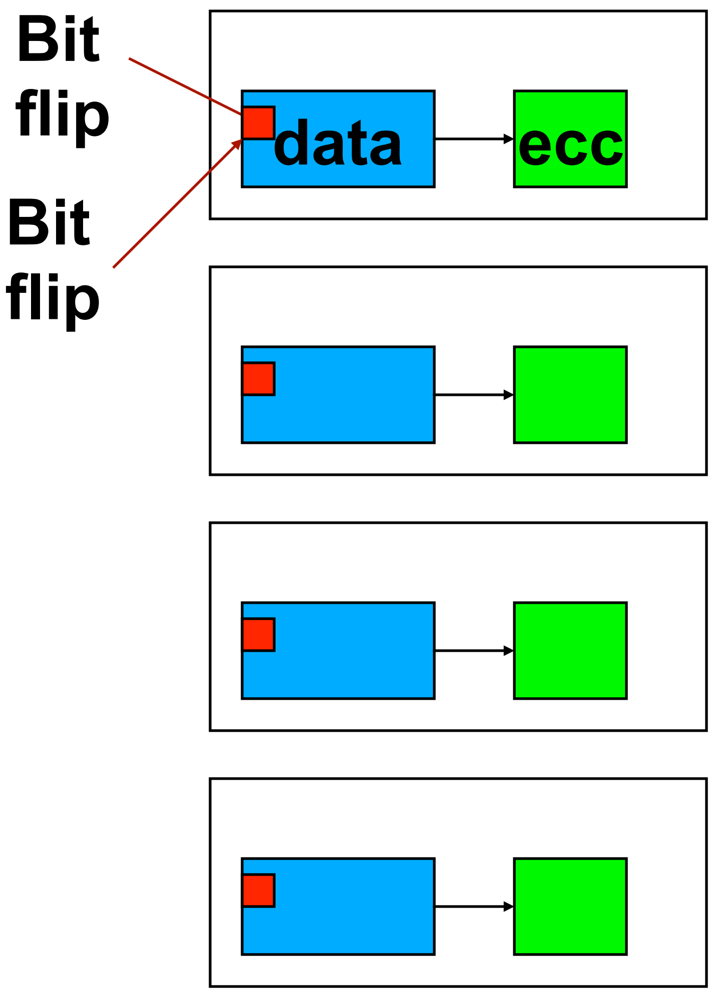
Example

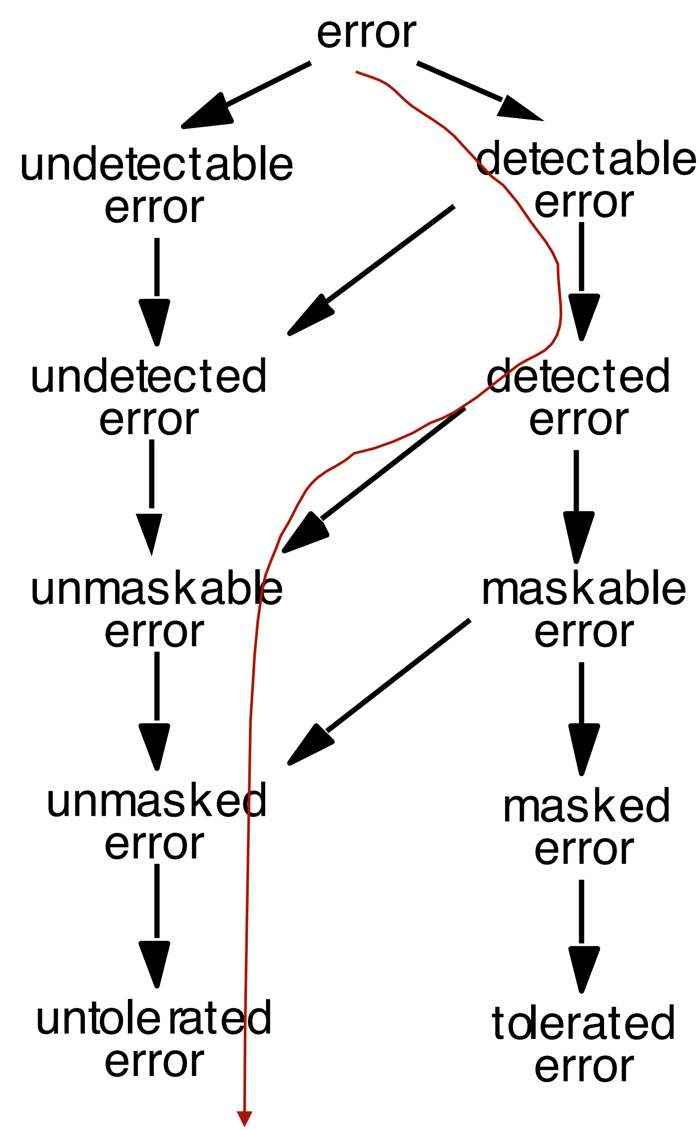




■ **Data needed by application**

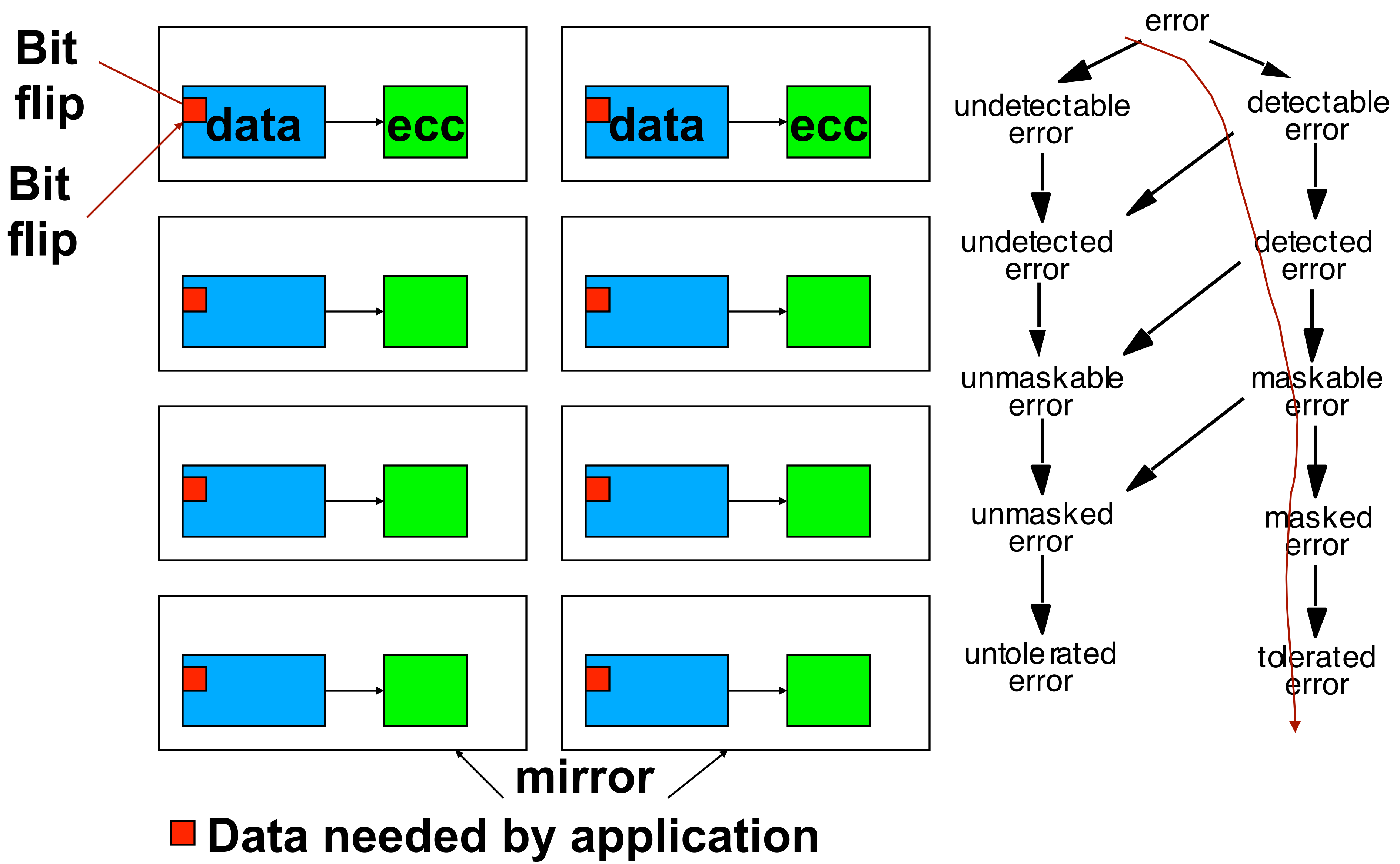
Example





Data needed by application

Example



Bit flip
Bit flip
data
ecc
data
ecc
mirror
Data needed by application
error
undetectable error
detectable error
undetected error
detected error
unmaskable error
maskable error
unmasked error
masked error
untolerated error
tolerated error

Reading

- Section 8.2-8.3