Handout #5

Chapter 3: Fault Tolerance

Overview

- What can go wrong
- How to fix it

What can go wrong

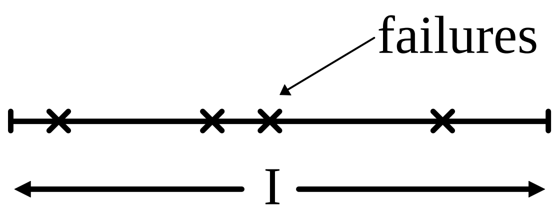
- hardware
- software
- environment
- people

⇒ wealth of data in textbook

TANDEM data [turn to textbook page 104, Figure 3.7]

- FT system; many hardware faults masked (only system outages reported)
- environment, operations outages underreported
- MTTF(89) = 20 years (1000 system years, 50 outages)
- MTTF(hardware) = 100 years (10 outages)

MTBF (mean time between failures) informal

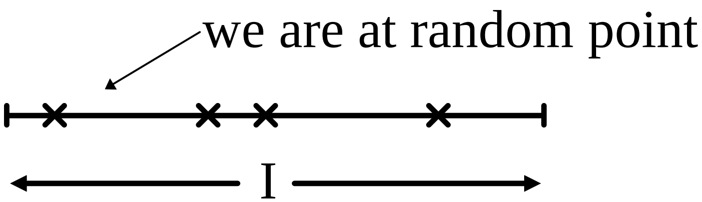


Frequency $= \frac{f}{I}$, where f = # failures, I = time

Note: Book calls this *probability* of failure.

Mean Time Between Failures $= MTBF = \frac{I}{f}$

<u>What about MTTF? (Mean Time To Failure)</u>



$\Rightarrow$ if fault distribution is *memoryless*:

$$\boxed{MTTF = MTBF = \frac{1}{freq} = \frac{I}{f}}$$

<u>Example:</u>

two types of faults: <u>r</u>ed and <u>b</u>lue
in 1000 years: 20 red, 30 blue

$\Rightarrow \ freq_r = \frac{20}{1000},\ freq_b = \frac{30}{1000},\ freq_t = \frac{50}{1000}$ where t: either r or b fails.

$\Rightarrow \boxed{freq_t = freq_r + freq_b}$ and $\boxed{MTTF_t = \frac{1}{freq_r + freq_b}}$

Example:

- memory system has 5 boards
- $MTTF_{board} = 10$ years
- system fails if any board fails
- what is the MTTF of the system?

$$MTTF_{system} = \frac{1}{\frac{1}{10}+\frac{1}{10}+\frac{1}{10}+\frac{1}{10}+\frac{1}{10}} \text{ years} = 2 \text{ years}$$

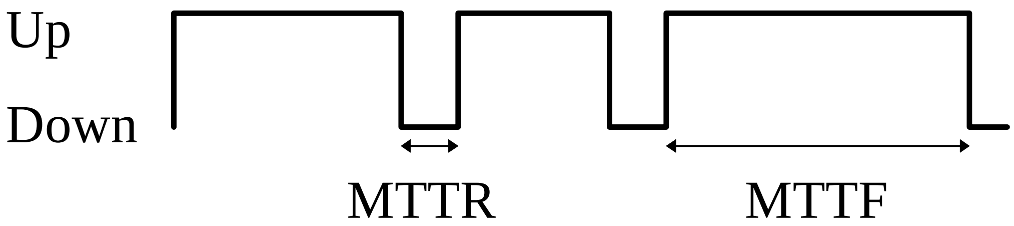
$$\Rightarrow \boxed{MTTF_{N\text{ equal things}} = \frac{MTTF_{each}}{N}}$$

[turn to textbook pages 104 and 106, Figures 3.8, 3.9]

Conclusions:

- need to eliminate high failure rates early & late in component life
- need to consider repairs

Repair Model



$$availability = \frac{MTTF}{MTTF + MTTR}$$

Comments to Figure 3.10 (see textbook page 107)

- disk reliability improving dramatically
  1985: MTTF = 8000 hours
  1990: MTTF=100,000 hours
- user expectations also increase!
  Also: number of disks growing (eg. RAID, large corporations) $\Rightarrow$ still have to take precautions
- should we protect against all failure types?
  eg: soft read error? Miss-corrected read?

How to fix it

Before we fix, we need to model failures:

- Events
  - Desired
  - Undesired
    - Expected
    - Unexpected

2 approaches:

- (A)
  - Columbus approach
  - Apollo approach
- (B)
  - Masking
  - Fail Stop (failsafe & recovery)

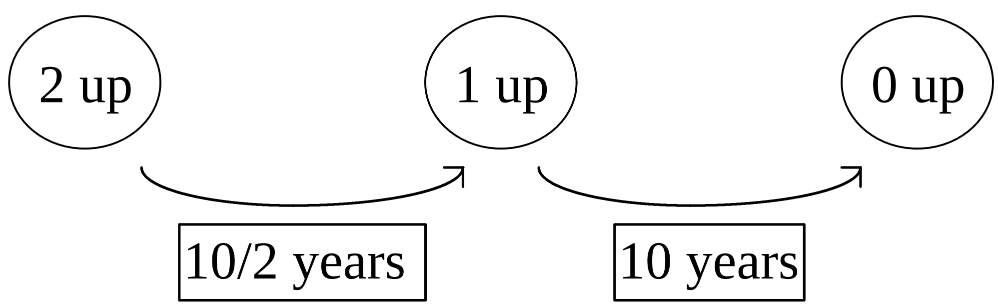
Options

	Mask	Failsafe&Recovery
Columbus		
Apollo		

⇒ Did you understand MTTF and availability computations for n-plex and failstop?

Example 1:

- 2 components A, B with MTTF = 10 years
- no repair
- if one fails, other continues

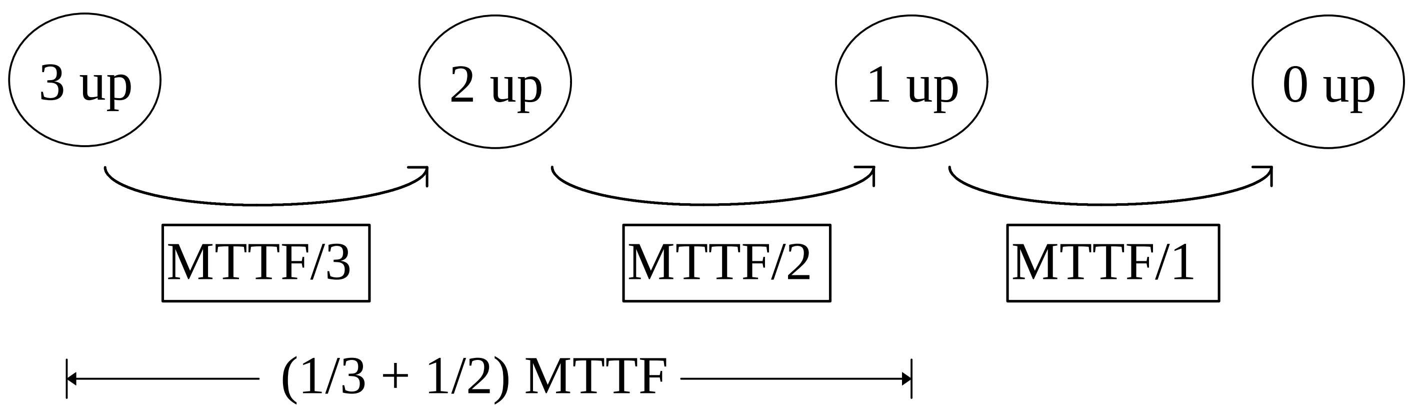


⇒ 15 years

Example 2

- 3 components, MTTF = 10 years
- TMR (Triple Module Redundancy) = system up until 2 fail
- voter does not fail

(a) no repairs



$$MTTF_{system} = \frac{5}{6} MTTF = 8.3 \text{ years}$$

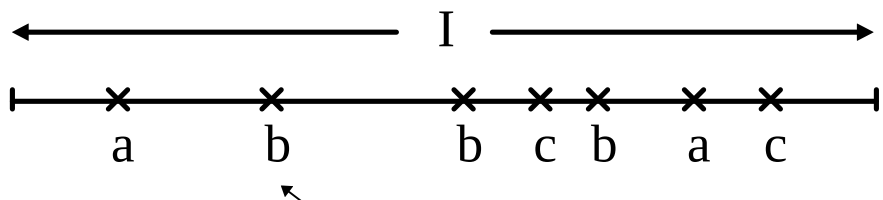
$\Rightarrow$ worse than 1 component!

$\Rightarrow$ why?

$\Rightarrow$ still useful? (see page 111-112)

(b) with repairs, MTTR = 1 day = 0.003 years

freq$_{\text{system-component}}$ = 3 freq$_{\text{component}}$



this b failure is only a system failure if both a and c down!

that two components are down happens with probability:

$$(1 - avail)^2 = \left[1 - \frac{MTTF}{MTTF + MTTR}\right]^2 = \left[\frac{MTTR}{MTTF + MTTR}\right]^2 \approx \left[\frac{MTTR}{MTTF}\right]^2$$

$$\Rightarrow \; freq_{system} = freq_{system-component}\left[\frac{MTTR}{MTTF}\right]^2 = 3\,freq_{component}\left[\frac{MTTR}{MTTF}\right]^2$$

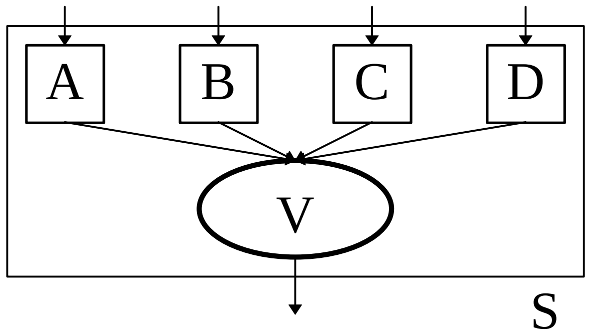
$$\Rightarrow \; MTTF_{system} = \frac{MTTF}{3}\left[\frac{MTTF}{MTTR}\right]^2 = \frac{10}{3}\left[\frac{10}{0.003}\right]^2 \text{ years} = 3.7 \cdot 10^7 \text{ years}$$

$\Rightarrow$ Real **BIG** Improvement!

Another Problem

- we have 4 components: A, B, C and D; MTTF=1 year; no repairs
- system is fail fast; voters do not fail

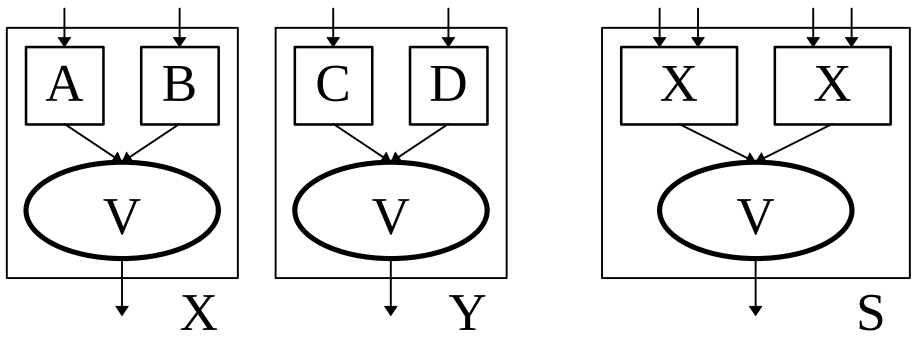
Option I



$$MTTF_S = \left(\frac{1}{4} + \frac{1}{3} + \frac{1}{2} + 1\right) \text{years}$$

$$= \frac{25}{12} \text{ years} = 2.08 \text{ years}$$

Option II

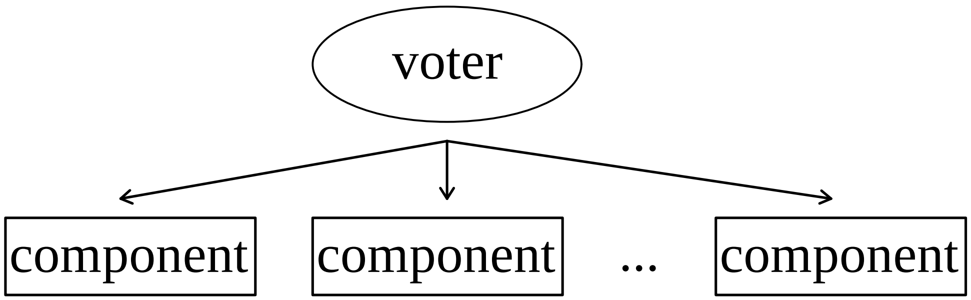


$$MTTF_X = MTTF_Y = \left(\frac{1}{2} + 1\right) \text{y} = 1.5 \text{ y}$$

$$MTTF_S = \left(\frac{1.5}{2} + 1.5\right) \text{y} = 2.25 \text{ y}$$

⇒ should be the same (system runs until all components down)
Problem: A,B,C,D are memoryless (components) but X,Y are not!

N-Plexing



Fail-Vote:

- majority of all components fail $\Rightarrow$ system fails

Fail-Fast:

- all components fail $\Rightarrow$ system fails
- assumes that voter can detect and ignore failed components

Summary so far

- What can go wrong
- How to evaluate reliability (MTTF, Availability)
- Overview of techniques for improving reliability

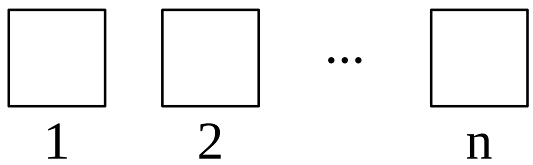
Next: How to make *storage* reliable.

Building highly available storage ("How to fix it!")

Overview

- Define model
- Make storage reliable
- Make more reliable

Disk Model



PAGE(i) : contents
STATUS(i) : good or bad (error code)

GET

(*status*, *block*) = GET(*address*)

GET(*i*) events

Desired:

IF status(i) = good THEN returns (good, page(i))

IF status(i) = bad THEN returns (bad, -)

Undesired:

Expected:

- status(i) = good BUT get(i) returns (bad, -) occurs at most k times in a row (soft read error)

Unexpected:

- status(i) = bad BUT get(i) returns (good, *)
- status(i) = good BUT returns (good, page(j)) where j ≠ i (undetected errors)
- get(i) never returns
- get(i) causes fire, data center burns up
- ...

Note:

Good Thing:

don't have to worry about undesired, unexpected events

Bad Thing;

have to make sure they are unlikely events (unlikely enough!)

Strategies:

(1) improve hardware (duplex, n-plec, ecc, ...)
(2) pray

PUT

`PUT(`*`address, block`*`)`

<u>**`PUT(`*`i,b`*`)` events**</u>

<u>Desired:</u>

- page(i) = b, status(i) = good

<u>Undesired:</u>

Expected:

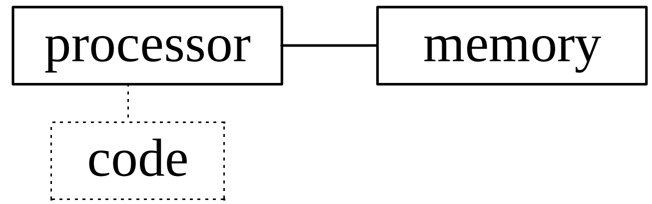
- page(i) not changed or status(i) = bad
  occurs at most k times in a row

Unexpected:

- take your pick!

<u>Assume:</u> no other undesired, expected events.
especially: disk pages do not go bad on their own (decay)

Processor Model



Desired events:

- code executed correctly
- memory never lost, corrupted
- code never lost, corrupted

Undesired events:

Expected:

- fail stop
- processor halts cleanly
- memory lost
- code not lost or corrupted
- after delay, execution resumes at recovery point

Unexpected:

- what have you.

Goal: recover from failures / errors

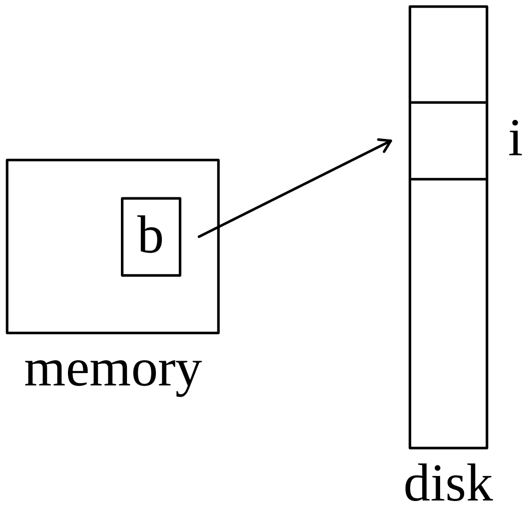
Example: soft read errors (i.e. page is good, but `GET` returns `STATUS=bad`)

```
(st, block) =  careful_get( i)
begin
  for j = 1 to (k+1) do begin
    (st, block) = get( i)
    if st = good then
      return ( st, block)
  end
  return (bad, -)
end
```

Progress

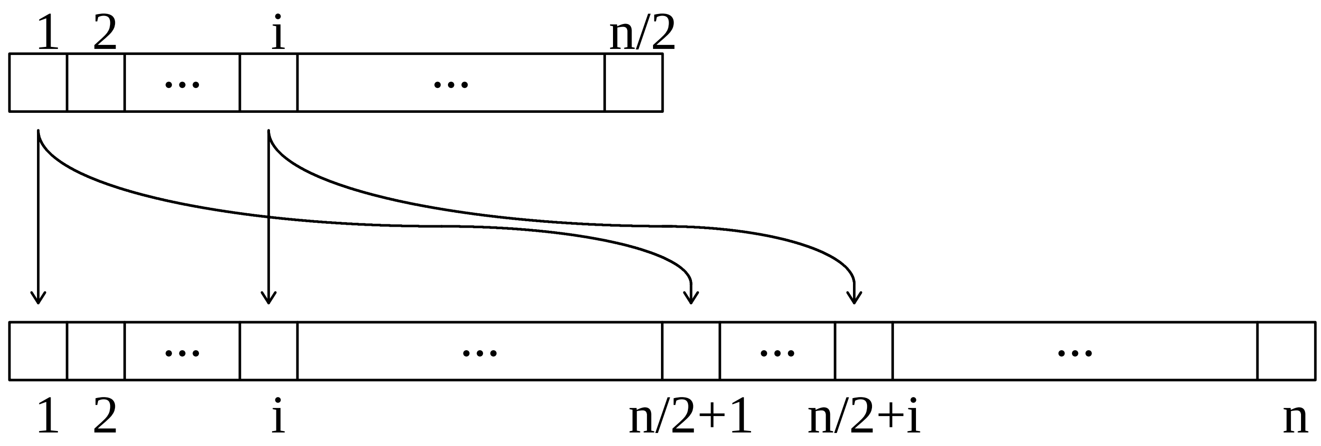
Have “eliminated” undesired, expected disk read event!

Can we do the same thing for write?



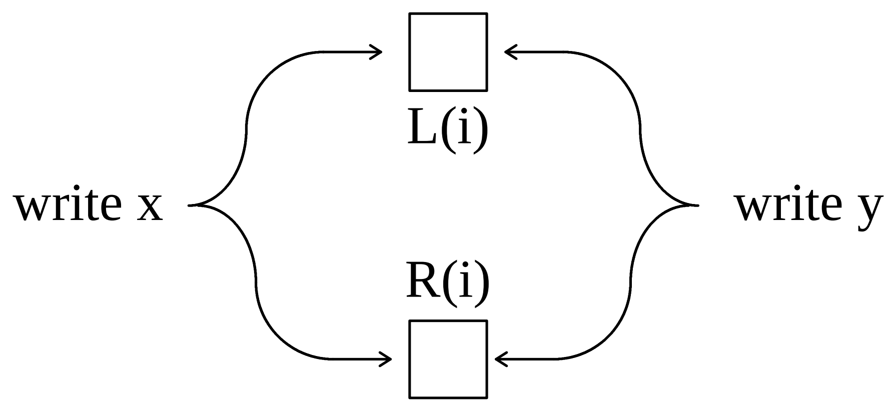
Processor failure during write:
- new value lost
- old value destroyed

Solution: **stable page**



Note:

- ignore concurrent operations:



$\Rightarrow$ need locking, see Chapter 7.

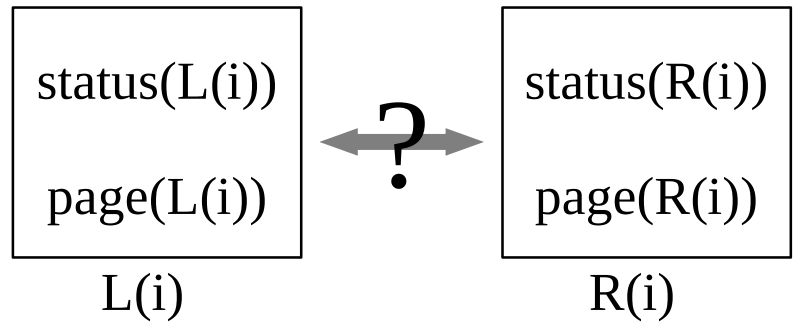
Careful and Stable Put

```
careful_put( i, block)
begin
   for j = 1 to (k+1) do begin
       put(i, block)
       (st, block') =  careful_get( i)
       if block=block' than return
   end
   // bad block - should not get here in our model
end
```

```
stable_put( i, block)
begin
  careful_put( L( i), block)
  careful_put( R( i), block)
end
```

Recovery

- need to clean up after failures




```
recovery()
begin
   (stL, blockL) =  careful_get( L( i) )
   (stR, blockR) =  careful_get( R( i) )
   if stL = good and  stR = bad then
       careful_put( R( i), blockL )
   if stL = bad and  stR = good then
       careful_put( L( i), blockR )
   if stL = good and
      stR = good and
      blockL !=  blockR then
       careful_put( R( i), blockL )
end
```

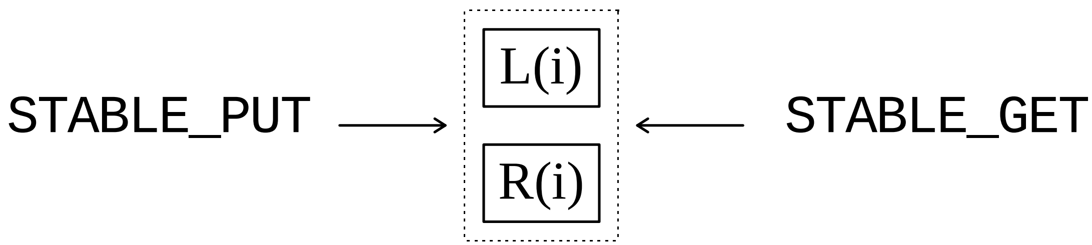
stable_get

- which page (L(i) or R(i)) do we read?

```
(st, block) =  stable_get( i)
   begin
       (st, block) =  careful_get( L( i) )
       return ( st, block)
   end
```

- it doesn't matter which one we read, they are the same
- might want to alternate reads for performance

What Have We Achieved?



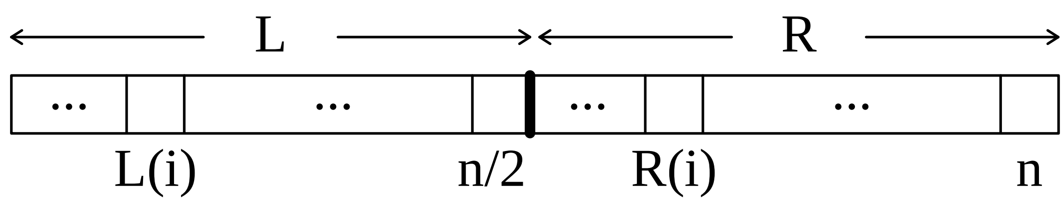
- Atomicity
- Durability

Decay

- decay = pages go bad on their own

Reasons for Decay

- dust
- `PUT(j)`
- cosmic ray
- head crash
- flood
- ...



Undesired, expected events:

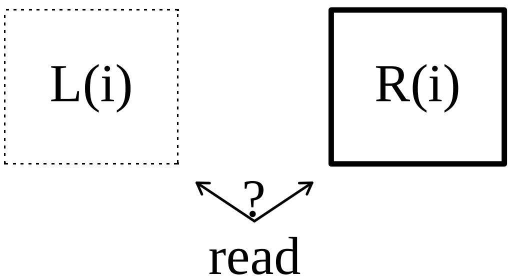
- set of L (or R) pages goes bad
- other pages not affected
- no other decays for at least T_d sec.

Decay Set:

- pages that can fault together, e.g.
  - Disk Drive
  - Disks in single building
  - ...

Coping with Decays

- put L(i), R(i) in different decay sets
- every T_d seconds: execute recovery procedure
- `stable_get` needs to cope with decay



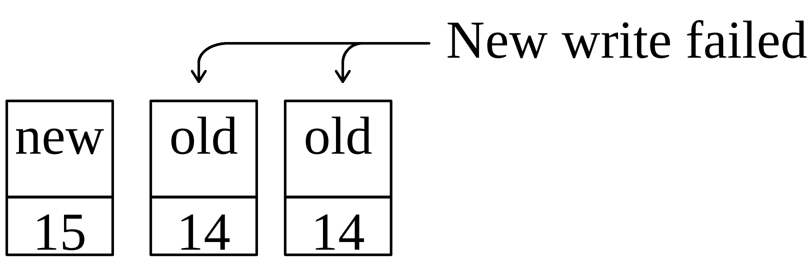

```
(st, block) =  stable_get( i)
begin
   (st, block) =  careful_get( L( i) )
   if st==good then return ( st, block)
   (st, block) =  careful_get( R( i) )
   return ( st, block)
end
```

Differences with textbook

- n-plex copies of each stable page

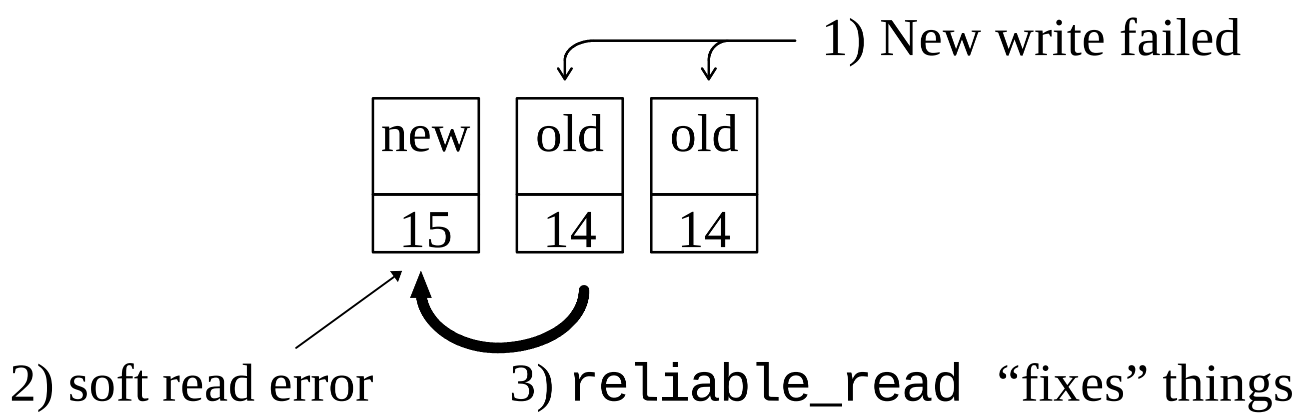
  p_1 p_2 p_3 ...

- no "careful" write: write n-plex copies without checking. Shotgun approach!
  - better performance
  - not as careful
  - needs version numbers



  - reads needs to read **all** n-plex copies (very expensive)
  - `reliable_read` does recovery: fixes bad or out-of-date pages it sees.

- ignores soft read error! (bug)

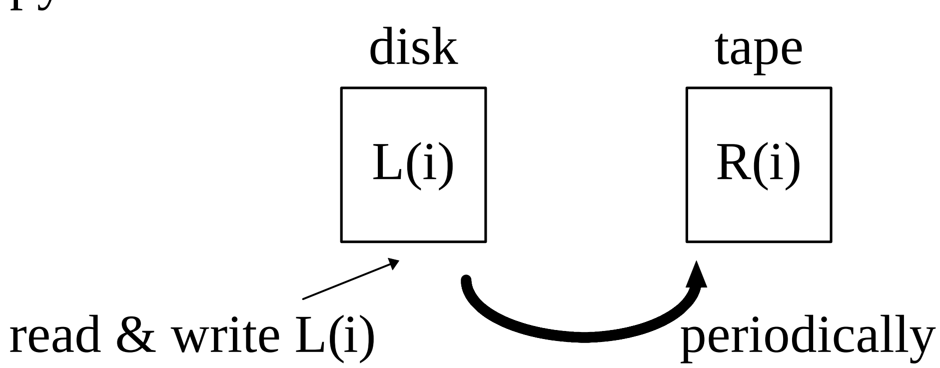


Summary

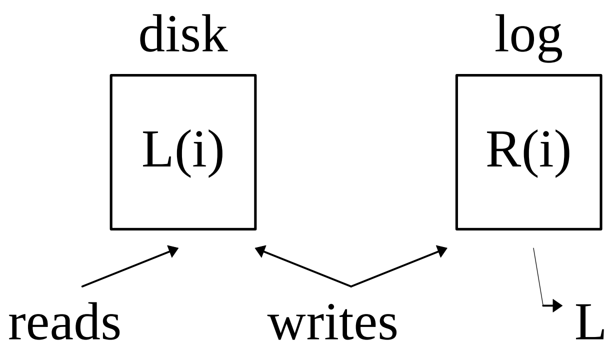
- textbook approach: efficient writes (e.g. good for log)
- stable get, put: efficient reads (Lampson, Sturgis)

Expensive?

- double, triple, ... storage
- expensive reads & writes
- recovery: read all data
- used in practice:
  logs, mirrored disks, control blocks, ...
- optimizations/ variations:
  - be sloppy, e.g. optimistic read, Sec. 3.7.2.5
  - one copy off-line



- one copy in log



Log: tape, disk, ...
- sequential access

- record active page ids in log

...	start write page i	...	start write page j	...	end write page j	...

crash

upon recovery:
- no need to check page j
- need to check page i

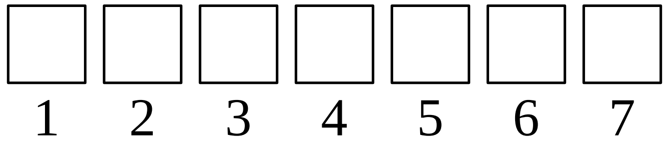
Final Note

- shown how to get durability and
- atomicity of disk write

BUT NOT

- atomicity of several writes
- isolation

Example Megatron 14X disk (14 pages)



7 logical pages

Transaction:

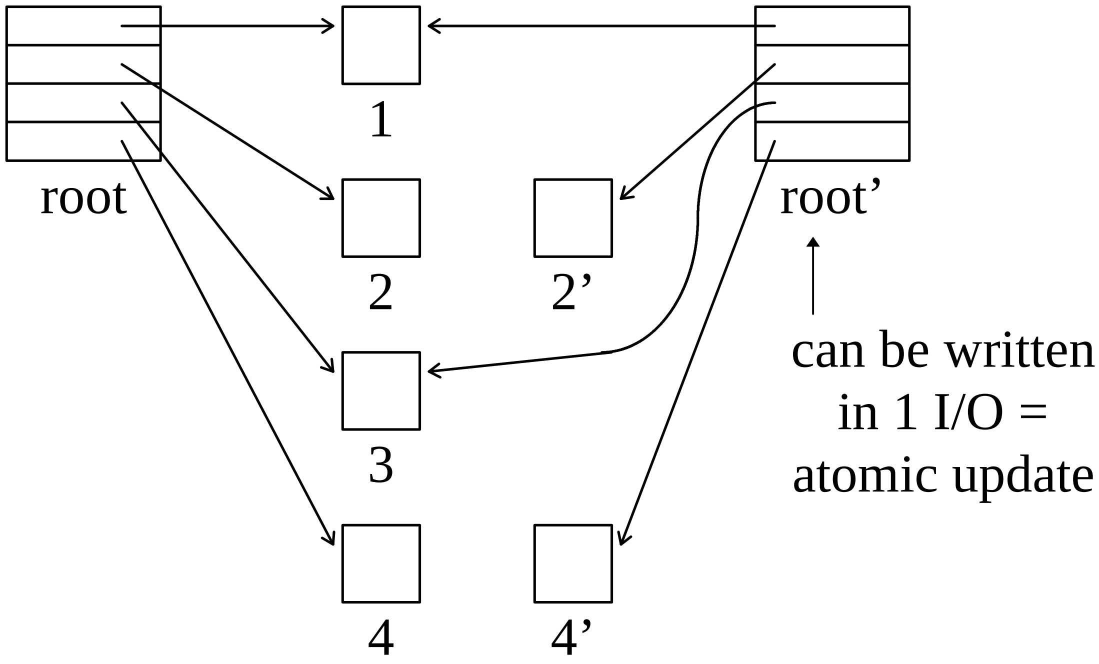
stable_put(2, ...)
stable_put(4, ...)

Solutions:

1) Logging

- write to log update to page 2
- write to log update to page 4
- commit
- stable_put(2, ...)
- stable_put(4, ...)

2) shadow pages



Notes:

- can have > 2 levels
- expensive for small changes: write new version, write root, de-allocate
- better for large changes, e.g. file system
- focus of book on logging

Final Final Note

highly available processes

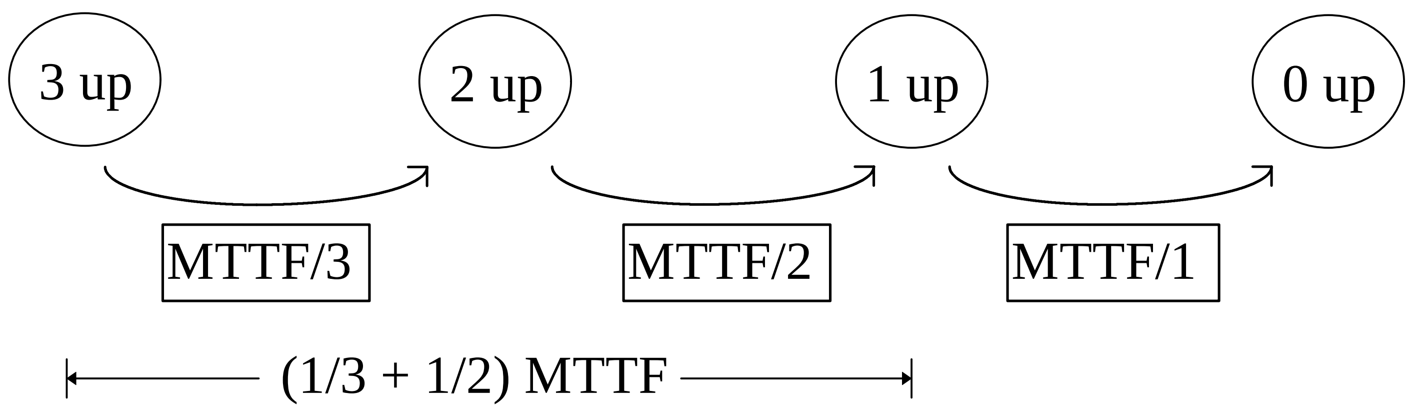
Read in textbook about

- checkpoint - restart
- process pairs
- persistent processes

Corrections to Example 2

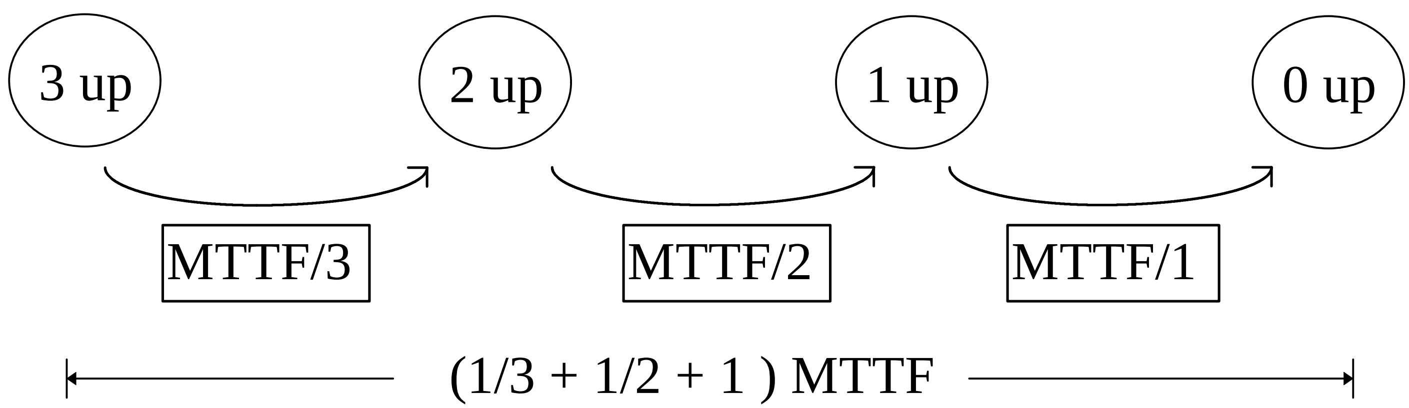
- 3 components, MTTF = 10 years
- voter does not fail
- fail vote: system up while at least 2 up
- fail fast: system up while at least 1 up

(a) fail vote, no repairs



$$MTTF_{system} = \frac{5}{6} MTTF = 8.3 \text{ years}$$

(b) fail fast, no repairs



$$MTTF_{system} = \frac{11}{6} MTTF = 18.3 \text{ years}$$

(c) fail vote, with repairs, MTTR = 1 day = 0.003 years

"one component is down" happens with probability:

$$P_1 = (1 - avail) = \left[1 - \frac{MTTF}{MTTF + MTTR}\right] = \left[\frac{MTTR}{MTTF + MTTR}\right] \approx \left[\frac{MTTR}{MTTF}\right]$$

The system becomes unavailable when
"one specific component already down and one or both of the others fail"
The probability of this event is:

$$P_2 = P_1 \cdot P(\text{either one or both other fail})$$

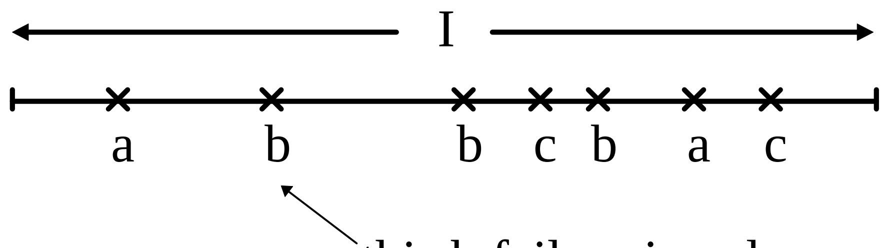
$$= (1 - avail) \cdot \left[\frac{1}{MTTF} + \frac{1}{MTTF} - \frac{1}{MTTF^2}\right] \approx \frac{MTTR}{MTTF} \cdot \left[\frac{2}{MTTF}\right] = \frac{2 \cdot MTTR}{MTTF^2}$$

$$\Rightarrow P_3 = P(\text{any one component down and one or both others fail}) = 3 \cdot P_2$$

$$\Rightarrow MTTF_{system} = \frac{1}{P_3} = \frac{MTTF^2}{6 \cdot MTTR} = 5555 \text{ years}$$

(d) fail fast, with repairs, MTTR = 1 day = 0.003 years

$$freq_{\text{system-component}} = 3\ freq_{\text{component}}$$



this b failure is only a system
failure if both a and c down!

"two components are down" happens with probability:

$$(1-avail)^2 = \left[1-\frac{MTTF}{MTTF+MTTR}\right]^2 = \left[\frac{MTTR}{MTTF+MTTR}\right]^2 \approx \left[\frac{MTTR}{MTTF}\right]^2$$

$$\Rightarrow\ freq_{system} = freq_{system-component}\left[\frac{MTTR}{MTTF}\right]^2 = 3\,freq_{component}\left[\frac{MTTR}{MTTF}\right]^2$$

$$\Rightarrow\ MTTF_{system} = \frac{MTTF}{3}\left[\frac{MTTF}{MTTR}\right]^2 = \frac{10}{3}\left[\frac{10}{0.003}\right]^2 \text{ years} = 3.7\cdot 10^7 \text{ years}$$