

# ***AFRAID* - A Frequently Redundant Array of Independent Disks**

---

*Stefan Savage*

Department of Computer Science and Engineering  
University of Washington

*John Wilkes*

Hewlett-Packard Laboratories

# Availability vs Performance

---

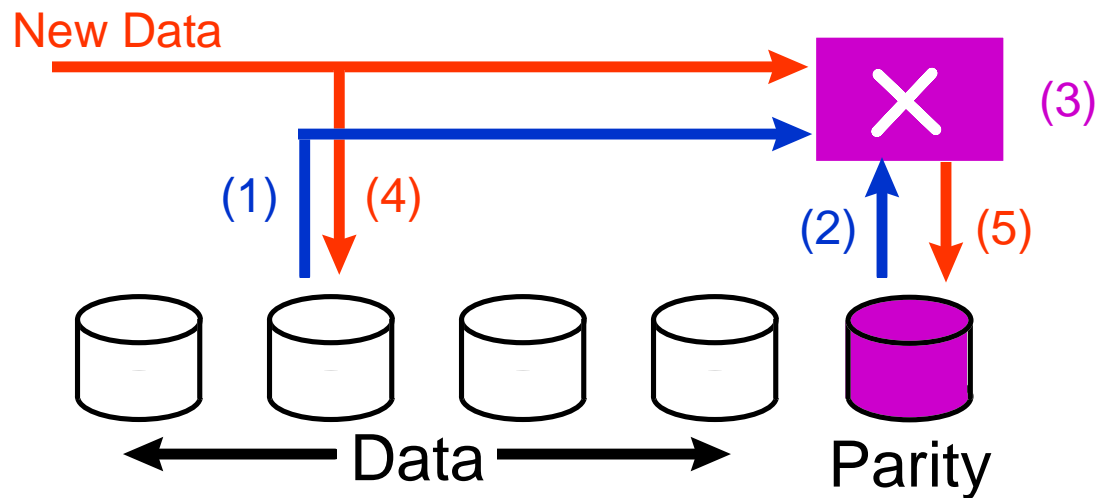
- Current arrays make fixed tradeoffs
  - *RAID 0*: very fast, but not allowed to crash
  - *RAID 1*: mediocre performance, good availability
  - *RAID 5*: prescription strength availability, but extremely poor small write performance
- ***AFRAID***
  - Dynamically trade availability for performance
  - *RAID 0* performance with > 50% *RAID 5* availability
  - 90% of *RAID 5* availability but with >40% better performance

# Cost of redundancy in *RAID 5*

---

*RAID 5* small-write protocol (4 I/O's)

- 1) Read old data
- 2) Read old parity
- 3) XOR new data with old data
- 4) Write new data
- 5) Write new parity



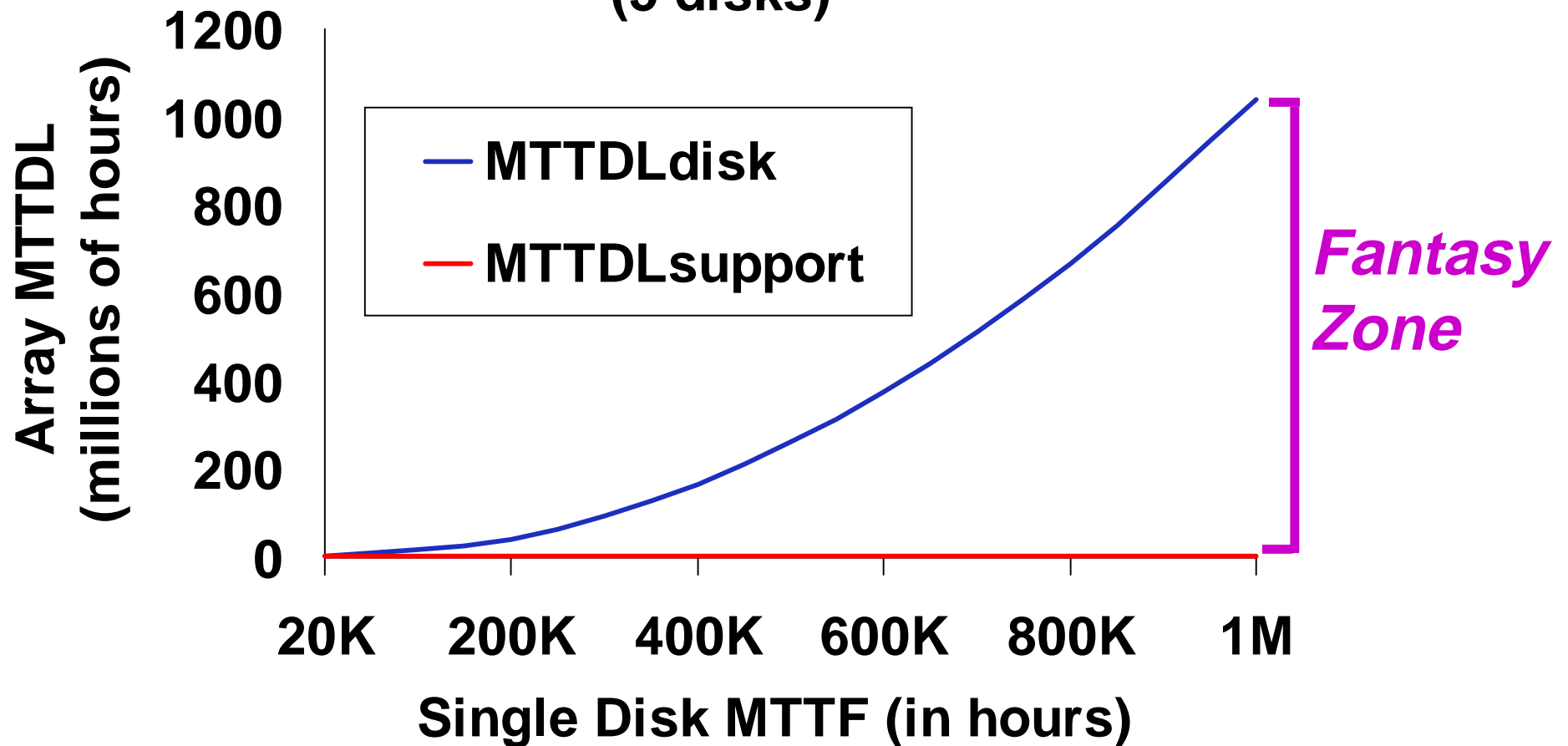
# Opportunity: Better disks

---

- 1980's disks: 20-30K hours MTTF
  - Original Berkeley *RAID* wanted to replace a single large disk (100K hours MTTF)
- Today's disks: 800K-1M hours MTTF
- Array availability is limited by support components
  - Memory, controllers, power supplies, etc...
  - Overall availability rate of 2M hours is very good!

# Opportunity: Useless redundancy

Sources of Failures in RAID 5 Disk Arrays  
(5 disks)



# Opportunity: Idle time

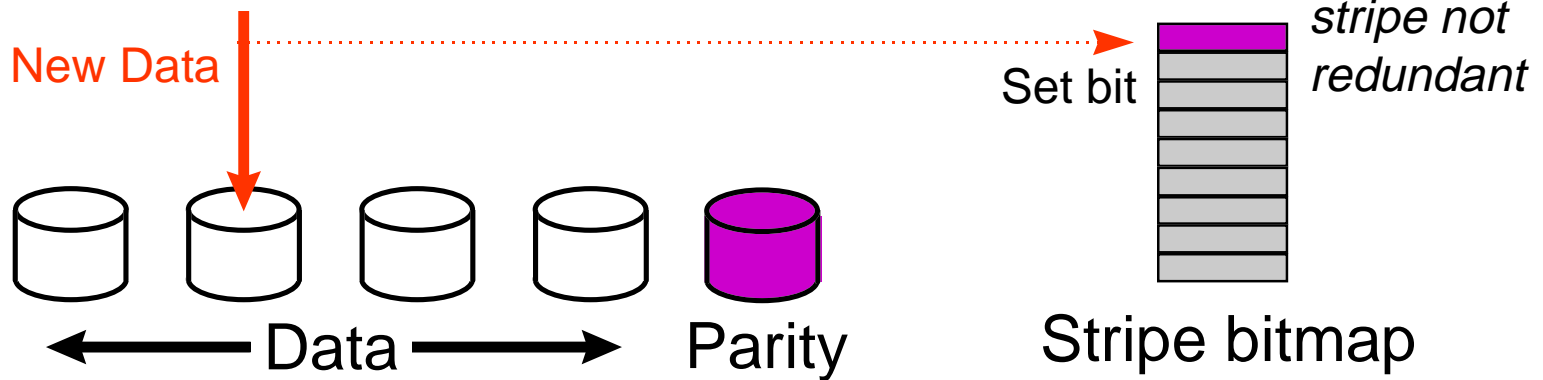
---

- There's lots of it in real life
  - Most of our traces have 90%+ idle time
  - Benchmarks aren't real life
- We can predict it fairly well
  - Golding et al, *Idleness is not Sloth*, USENIX '95
- We could use it for something...

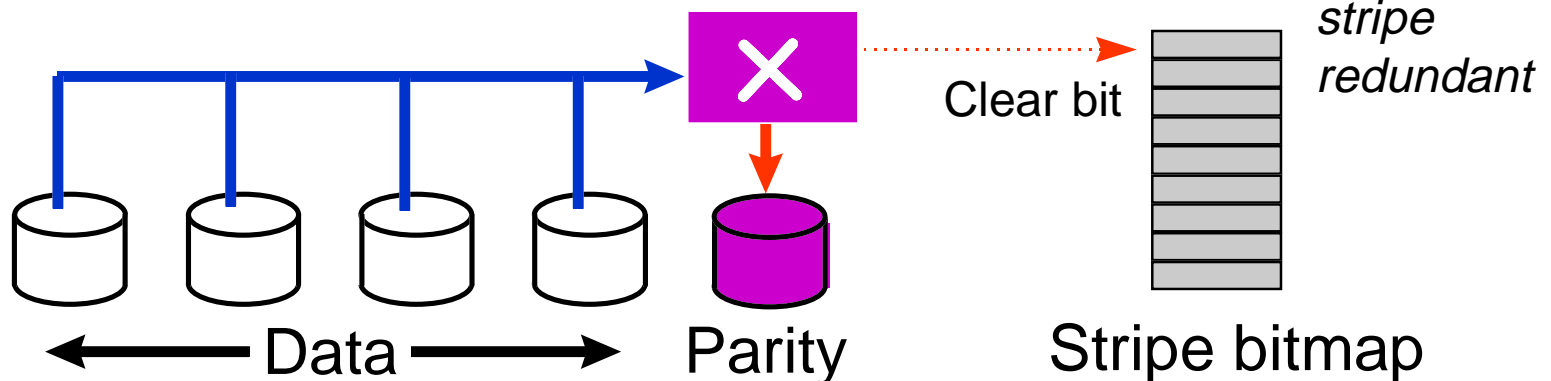
# AFRAID strategy

---

## Small-write, in foreground



## Parity-update, in background



# *AFRAID*: More details

---

- **Stripe bitmap**
  - Indicates that a stripe doesn't have valid parity
  - Stored in NVRAM
  - Array still functions correctly even if bitmap is lost
- **Parity updates**
  - Batched together for efficiency
  - Unprotected data bounded by rebuild rate and frequency of parity rebuild

# Controlling availability

---

- Limit amount of unprotected data
  - Start parity update when too much data is unprotected
  - Start parity update if data is unprotected too long
- Revert to *RAID 5* mode
  - When computed MTTDL is too high
  - User supplies desired MTTDL threshold
  - Any extra availability is translated into improved performance

# Key points

---

- Just one I/O per small write
- Array temporarily unredundant during bursts of writes
- Parity update is expensive, but can be “free” to user if it happens in idle time
- By reverting to *RAID 5* behavior we can control overall availability
- User can specify MTTDL

# Evaluating *AFRAID*

---

- Performance
  - Up to the level of *RAID 0*
- Availability
  - Up to the level of *RAID 5*
- Effectiveness of tradeoff
  - Real performance improvements from sacrificing small fraction of availability

# Simulating *AFRAID*

---

- Event driven simulation
  - *RAID 0*, *RAID 5* and *AFRAID*
  - Idleness predicted by 100ms idle period
  - write-through caches
- Used real workloads
  - hplajw, snake, cello - software development
  - netware - synthetic benchmark
  - ATT, as400-x - database workloads

# Simulation assumptions

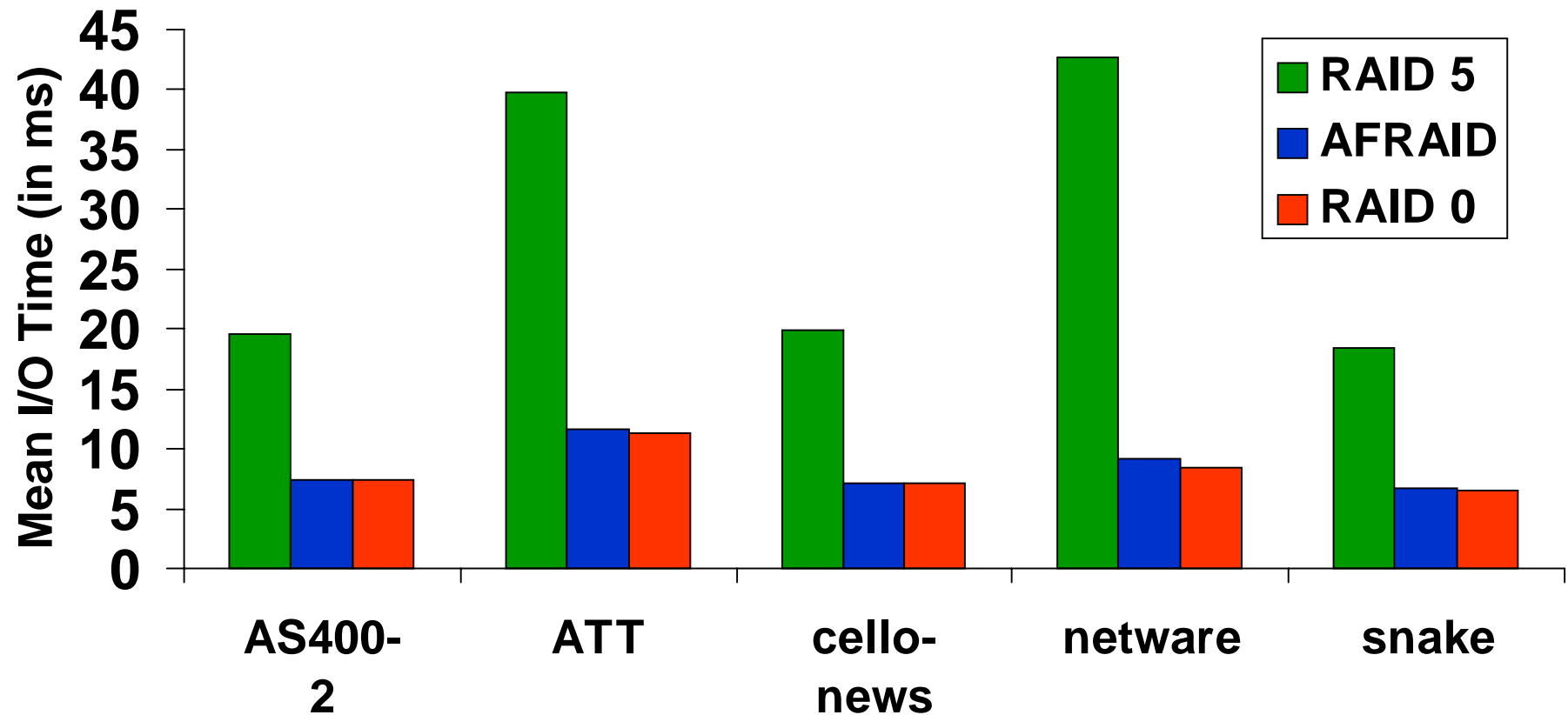
---

MTTF <sub>disk</sub>	1M hours
MTTDL <sub>support</sub>	2M hours
% disk failures predicted	50%
MTTR	48 hours
Stripe unit size	8KB
Size of disk	2GB

# Performance

---

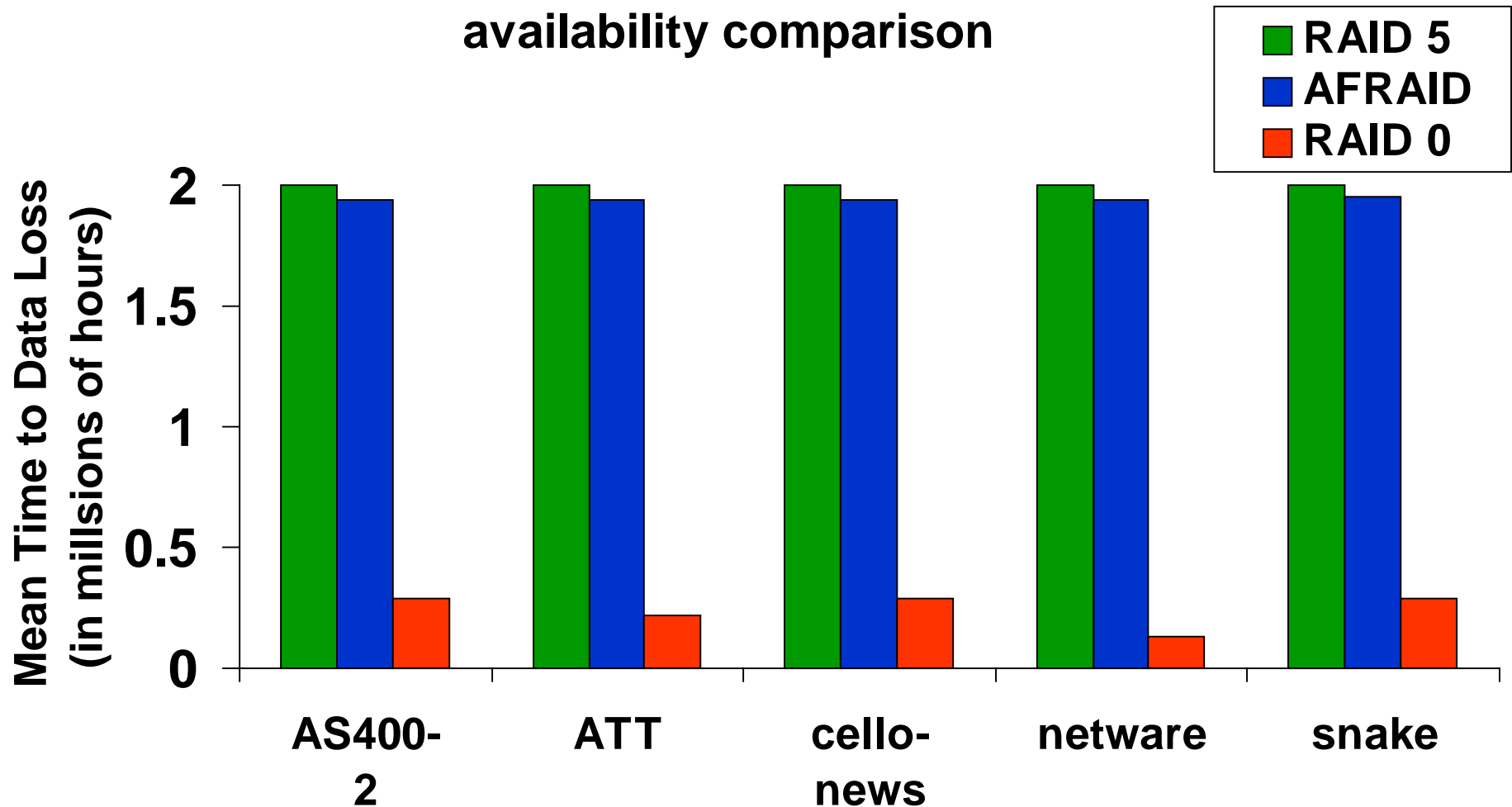
RAID 5, RAID 0, and AFRAID-MTTDL-0 performance comparison



# Availability

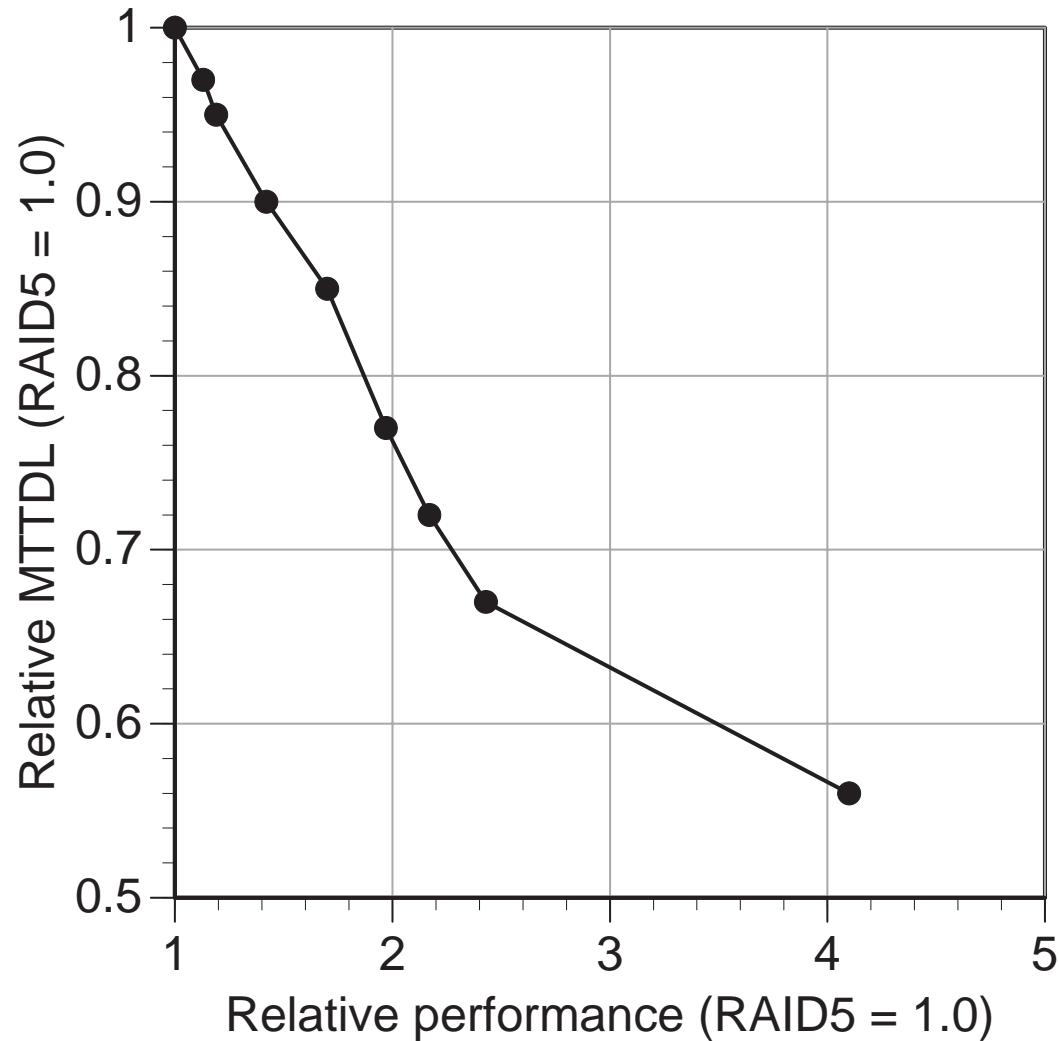
---

RAID 0, RAID 5 and AFRAID-MTTDL-64  
availability comparison

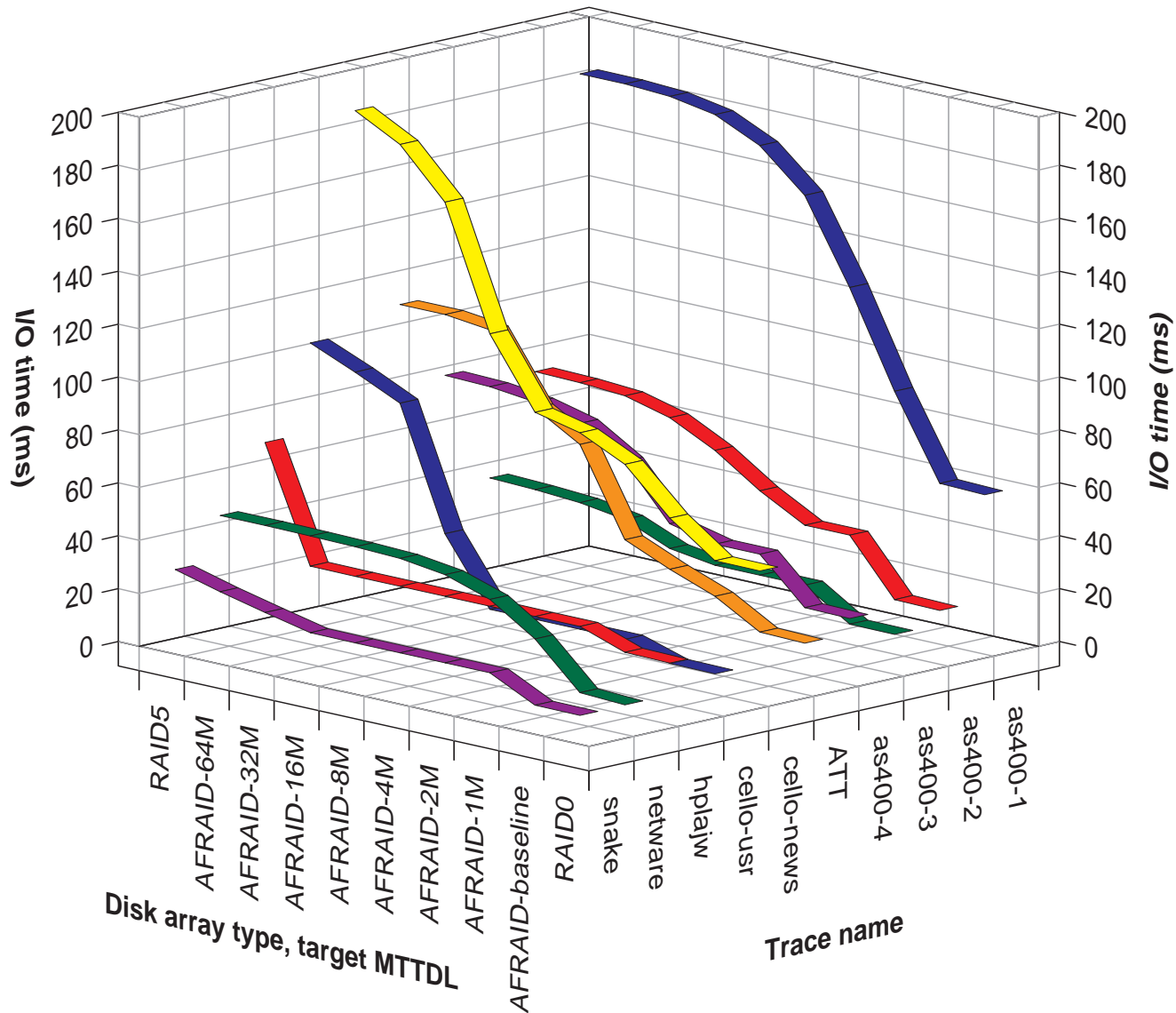


# Availability/Performance tradeoff

---



# Availability/Performance tradeoff



# Conclusions

---

- Availability is an end-to-end issue
  - don't be fooled by the fantasy zone
- Idleness happens
  - don't be fooled by benchmark loads
- *AFRAID*
  - Allows user to specify a required availability
  - Will turn any excess availability into performance

# Related Work

---

- Parity Logging (*Stodolsky et al*)
  - Defer cost of parity update by logging XOR of old data and new data
  - Replay log file later to update parity
- Floating parity (*Menon et al*)
  - Make parity update cheaper, by putting parity in a rotationally-nearby slot
- *Cormen et al*
  - Proposed that files be unredundant when created and made redundant under application control

# Array Parameters

---

- Number of disks
  - ATT used 5 disks, netware used 8
  - All other traces used 4 disks
- Disks
  - HPC3325 2GB disks (5400 RPM)
  - 9ms avg seek
  - 3.2-5.1 MB/s transfer rate
- Cache
  - 256KB write staging area (Write through)
  - 256KB read cache

# Why not just use NVRAM?

---

- Cost

- Good NVRAM (integral lithium cell) is roughly \$350/MB
- Fairly reliable

- Reliability

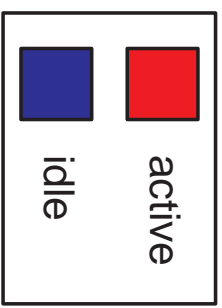
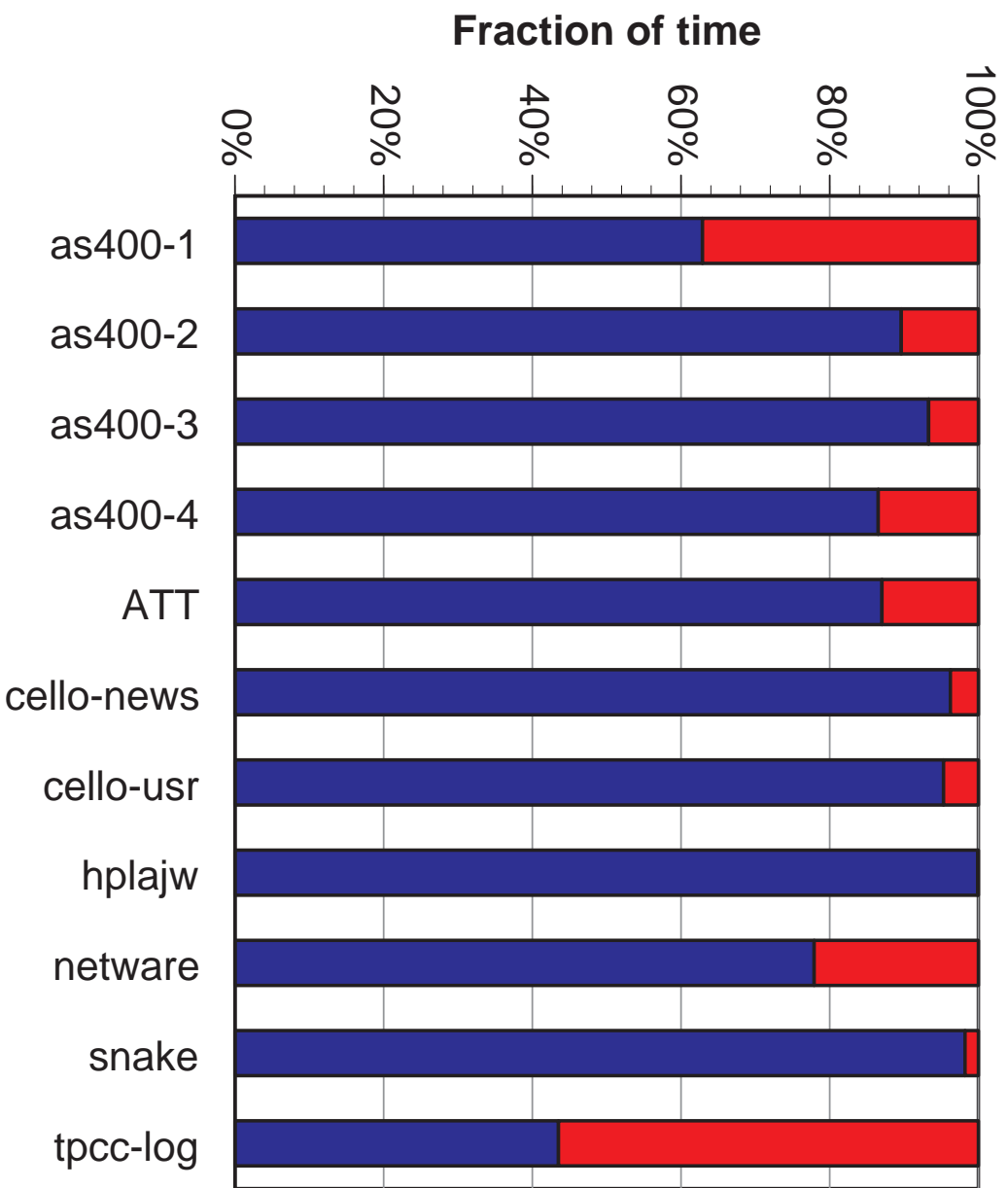
- Popular NiCd battery-backed SRAM is much less expensive (and more common), but...
- Much less reliable
- Popular PrestoServe card has MTTF of only 15k hours

# NVRAM + *AFRAID*?

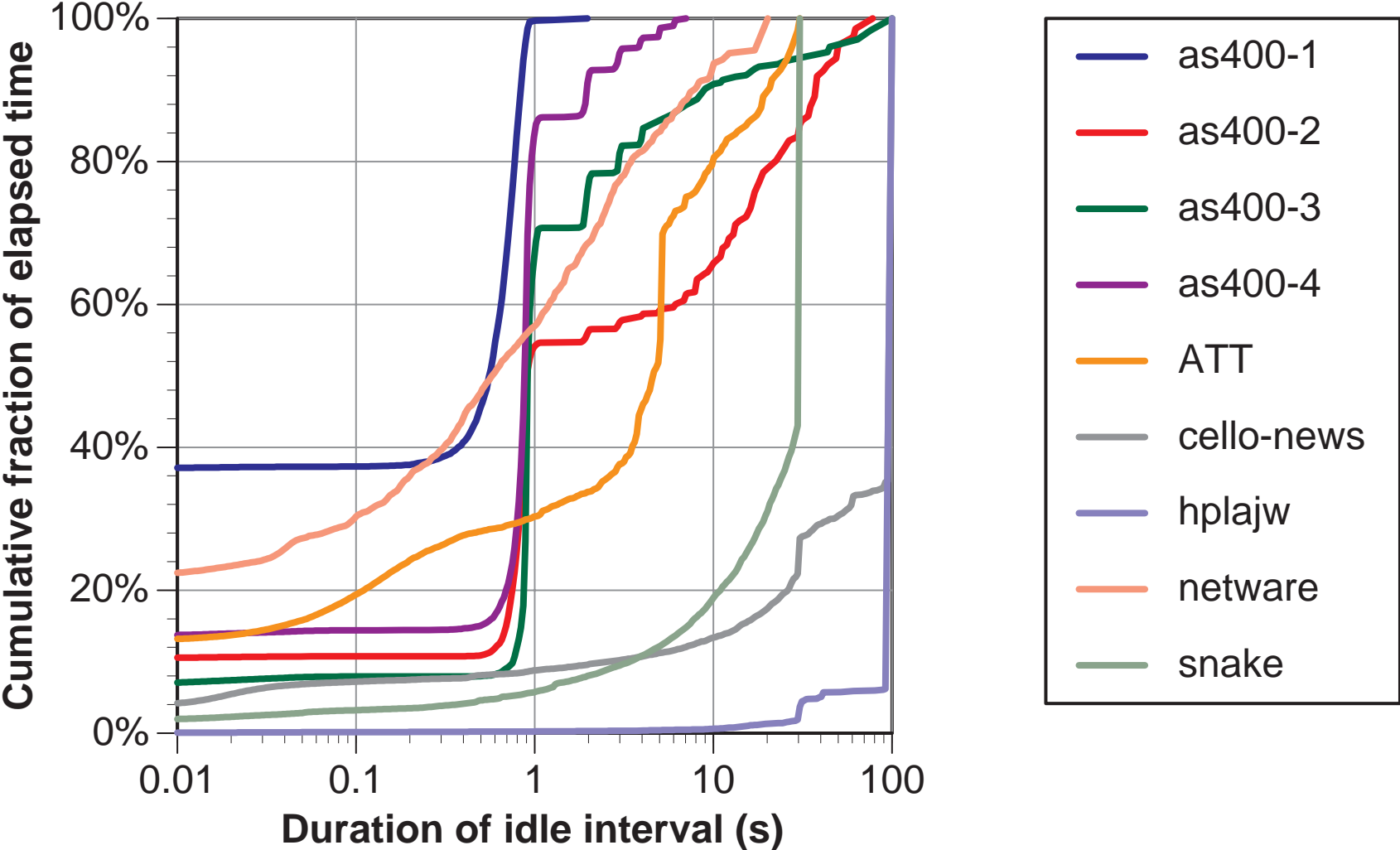
---

- Small NVRAM cache
  - NVRAM catches frequent small bursts
  - AFRAID absorbs rarer, larger bursts
  - AFRAID increases NVRAM-flush rate
- Reduces MTTDL by reducing the amount of unprotected time

# Idle Time

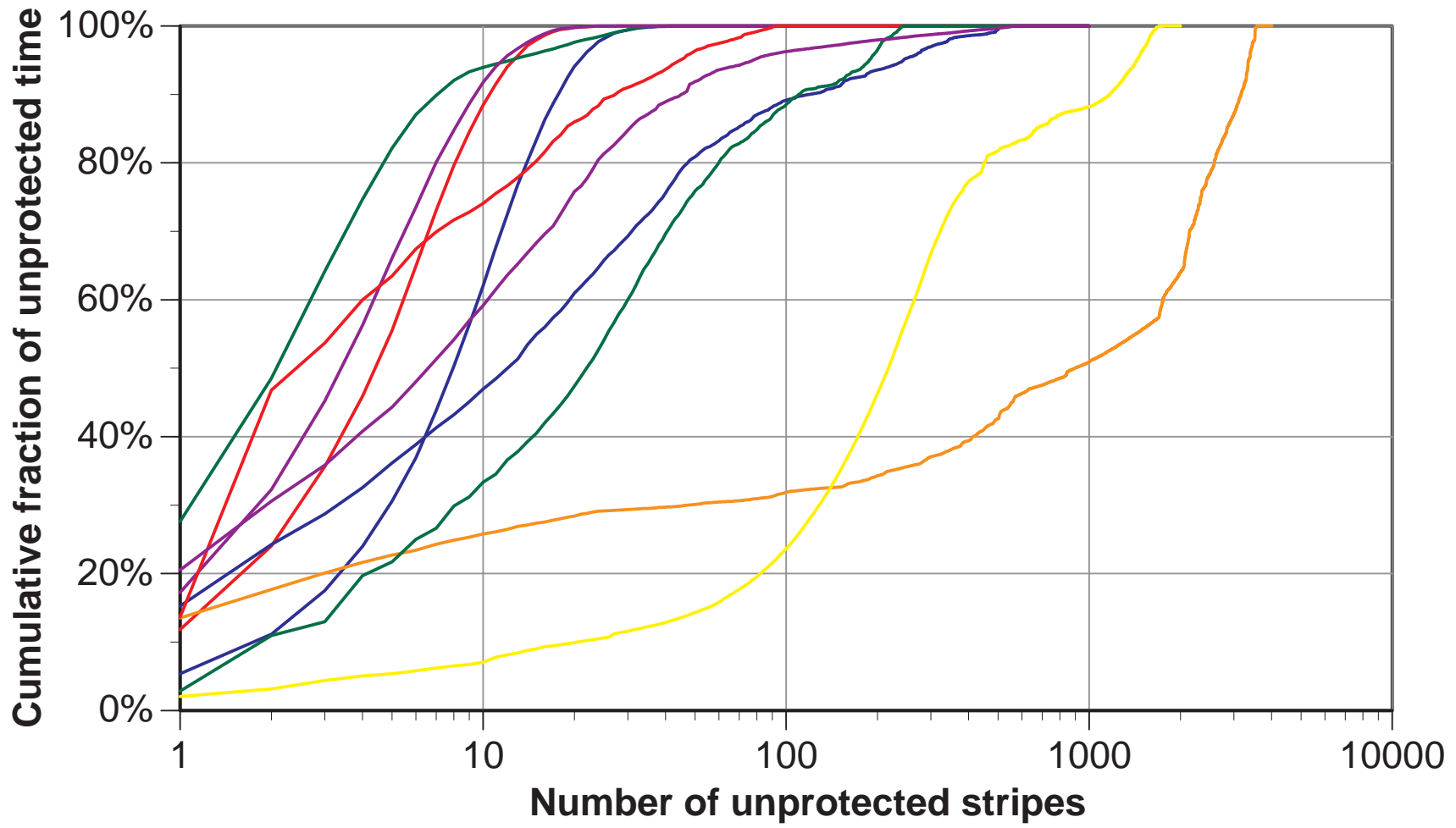


# Idle Period Length



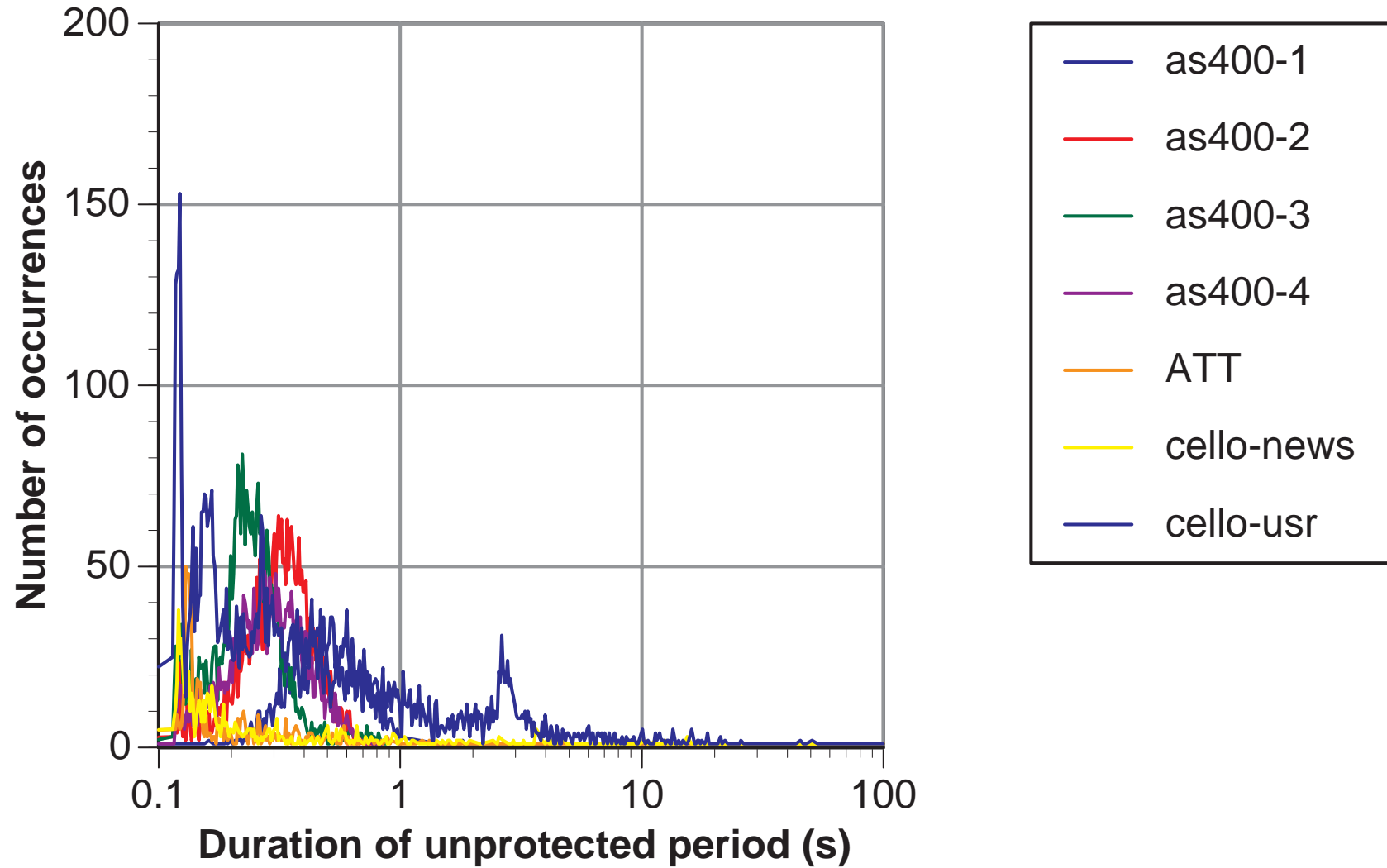
# Amount of unprotected data

---



# Duration of unprotected periods

---



# AFRAID availability theory

---

- $MTTDL_{RAID} = \text{time to lose 2 disks}$   
 $= (MTTF_{\text{disk}})^2 / (N(N+1) \times MTTR)$
- $MTTDL_{AFRAID} =$   
MTTF<sub>disk</sub> for the % time when the array has  
unredundant data, and MTTF<sub>RAID</sub> for the % time  
when all data is redundant
- $MTTDL_{\text{overall}} =$   
 $= 1 / (1 / MTTDL_{AFRAID} + 1 / MTTDL_{\text{support}})$

# Causes of data unavailability

---

- Double disk failure (like RAID 5)
- Single disk failure when data is unprotected
- Failure of support components (2M hours)