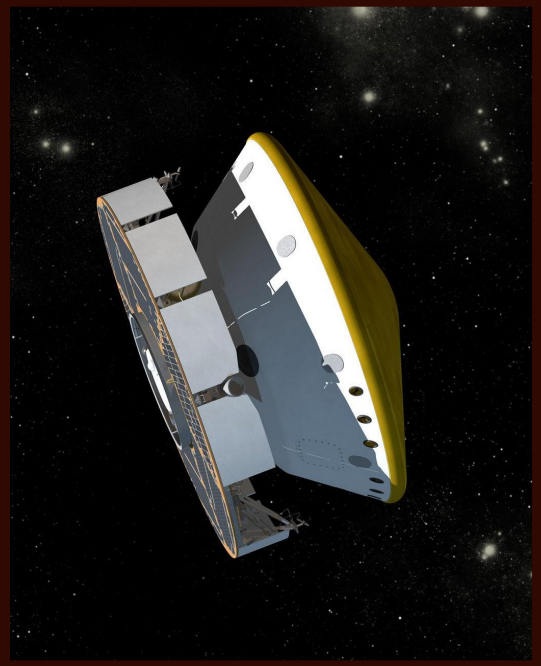
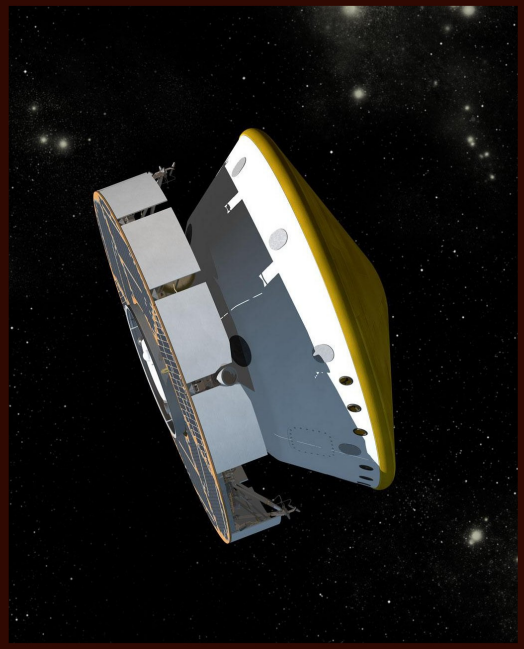


Managing data for Curiosity, fun and profit

Rajeev Joshi
NASA / Jet Propulsion Laboratory,
California Institute of Technology

3 missions in one



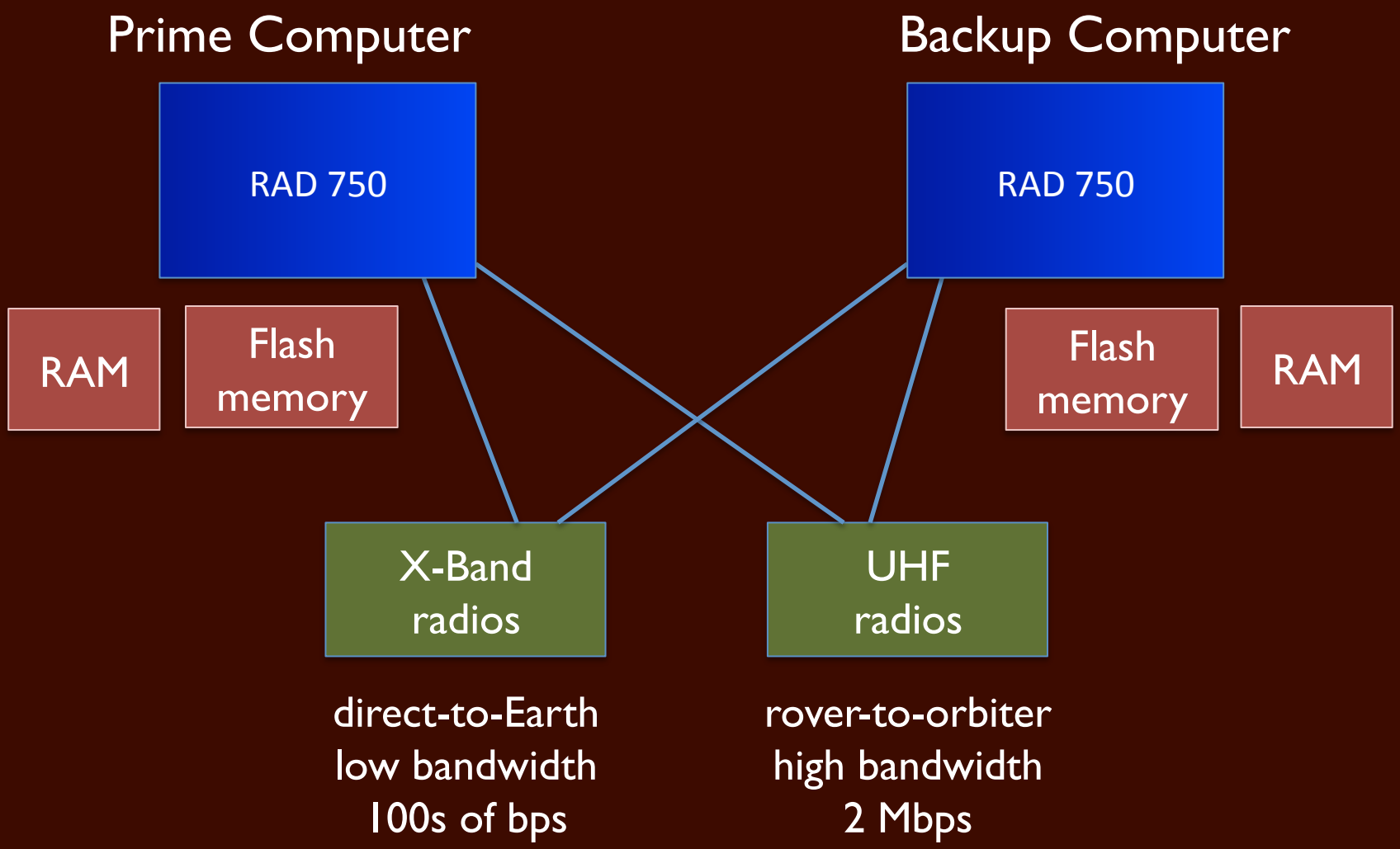
Cruise

EDL

Entry, Descent & Landing

Surface

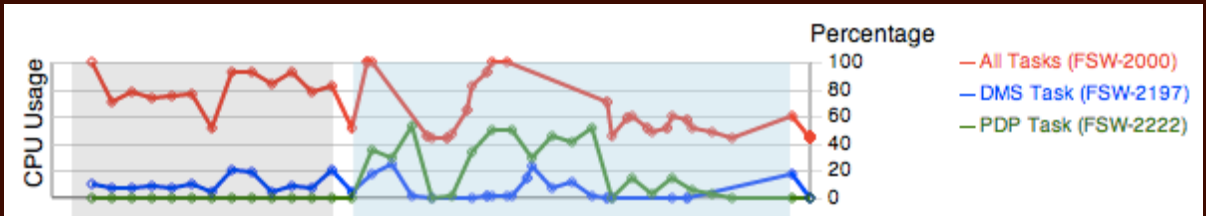
Avionics


Prime Computer
Backup Computer
RAD 750
RAD 750
RAM
Flash memory
Flash memory
RAM
X-Band radios
UHF radios
direct-to-Earth
low bandwidth
100s of bps
rover-to-orbiter
high bandwidth
2 Mbps

“Realtime” telemetry

Streamed continuously whether or not ground is listening

Periodic measurements (“EHA”)



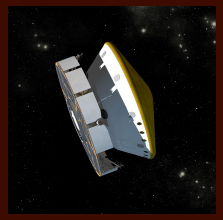
Discrete event log (“EVRs”)

ERT	LMST	SEVERITY	MESSAGE
2013-354T09:44:00	SOL-477M13:01:03	ACTIVITY_HI	Preparing telecom subsystem for UHF Comm Window W34772
2013-354T09:44:01	SOL-477M13:01:07	COMMAND	Dispatched command to turn on prime Electra radio
2013-354T09:44:02	SOL-477M13:02:37	DIAGNOSTIC	Successfully turned on prime Electra
2013-354T09:44:03	SOL-477M13:02:40	ACTIVITY_LO	Starting packetization of data products for comm window W34772
2013-354T09:44:04	SOL-477M13:07:33	WARNING_LO	DMS task did not report within threshold of 60 seconds
2013-354T09:44:05	SOL-477M13:07:36	ACTIVITY_LO	DMS task reported 3 seconds late to health ping request
2013-354T09:44:06	SOL-477M13:12:40	ACTIVITY_LO	Finished comm window W34772;reestablishing background mode

“Recorded” telemetry

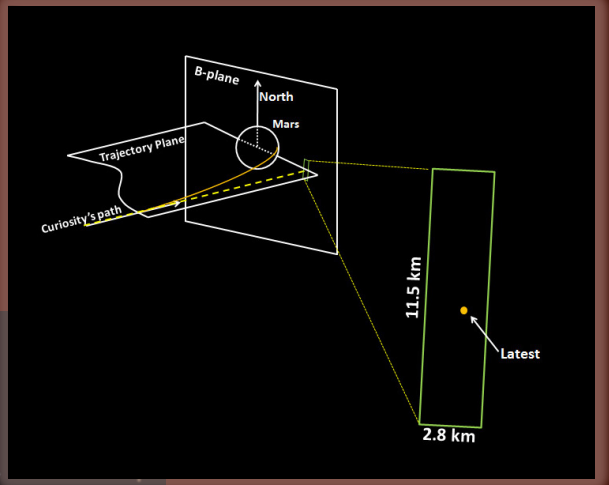
Realtime telemetry saved to nonvolatile memory
Sent only when requested by ground

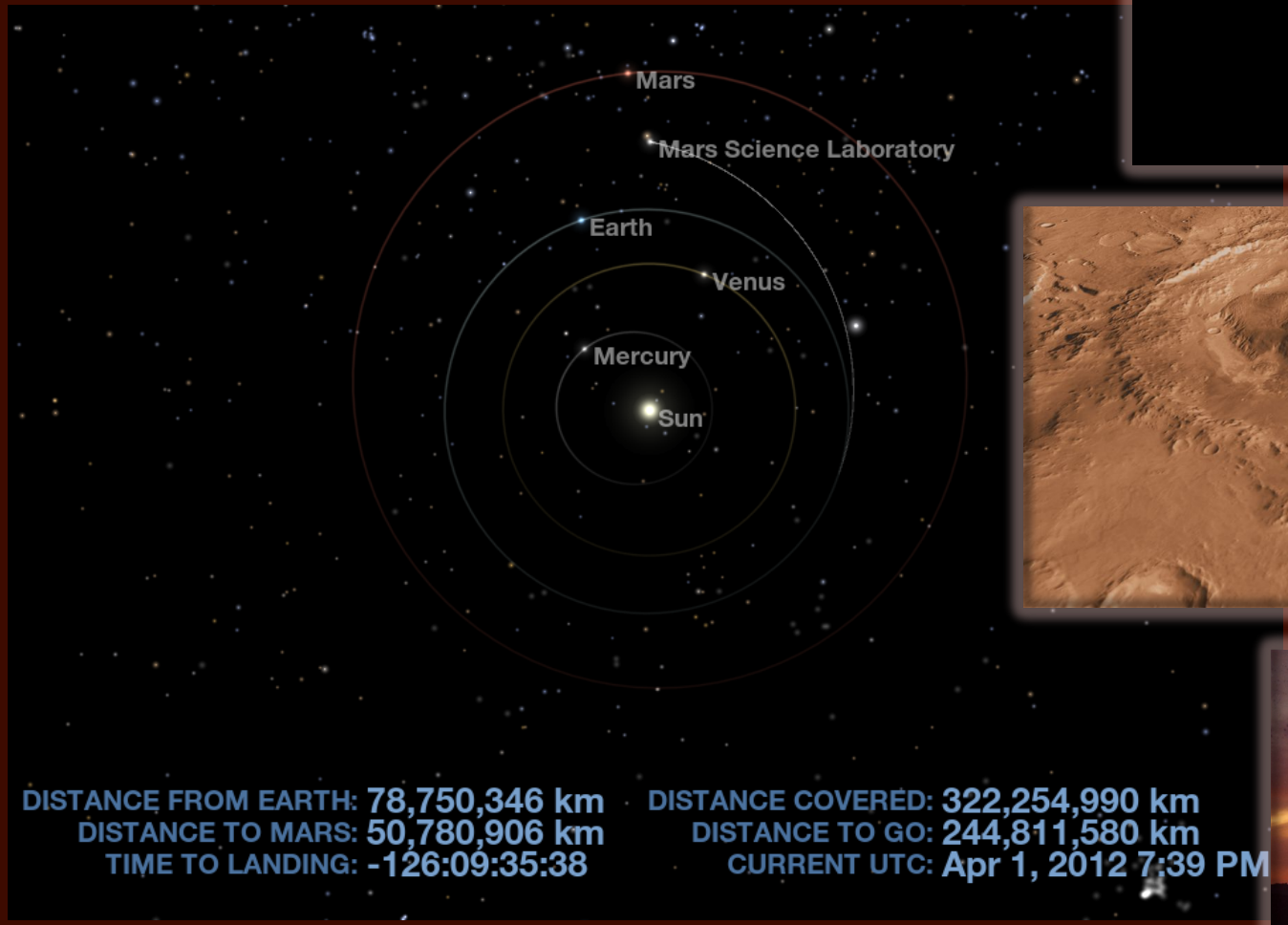
“Data products” containing
detailed engineering data (e.g., battery charge history)
science observations (images, spectra, …)

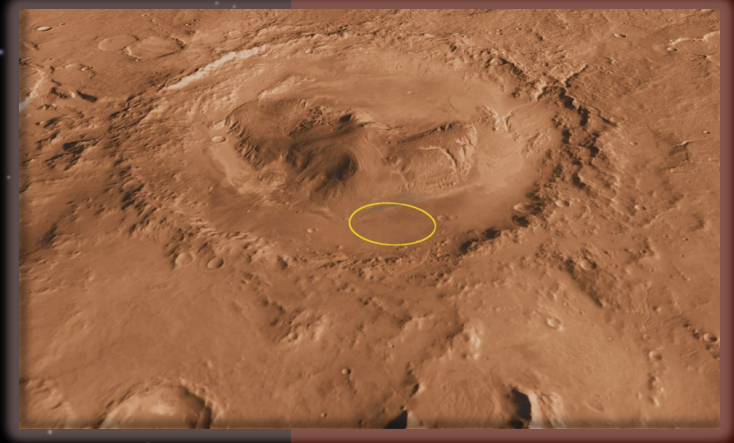


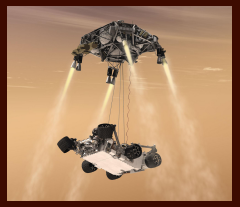
Cruise

8.3 months



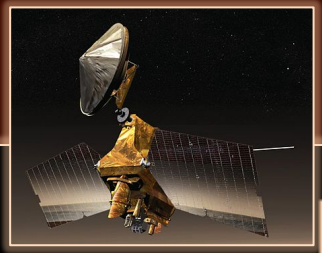


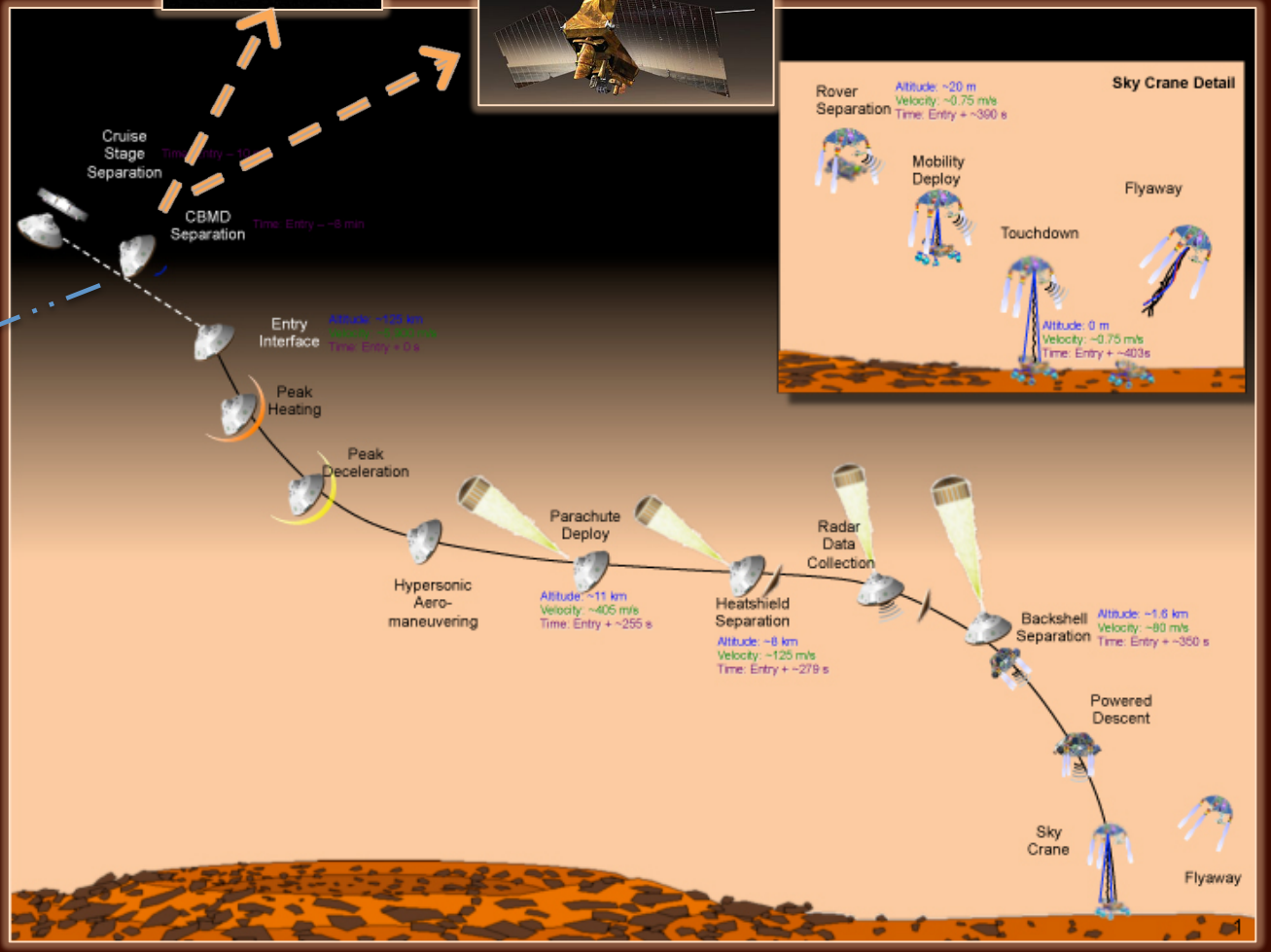




EDL

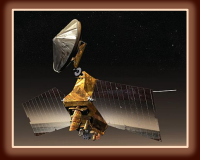
7 minutes

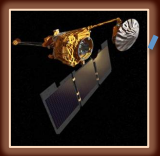


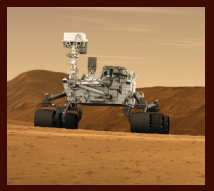


Surface

>2 years







10:00 16:00 02:00 10:00

Data Management Needs

Cruise

- downlink data in priority order when requested
- support on-demand transfer of data from backup computer
- support commanded reprioritization, retransmission and deletion

EDL

- support high-rate, thread-safe collection of data
- stream data to orbiters during descent
- commit data to flash before surface transition

Surface

- support multiple wakeup/shutdown cycles per sol
- retrieve data from flash in time for comm windows
- support “virtual data products”

Challenges

Processor speed	133 Mhz
Main memory	128 MByte (same as MER)

Over 65 Level-3 and Level-4 functional requirements

Support upto 300,000 data products
- fast retrieval of data in priority order for downlink
- support efficient catalog update (e.g., reprioritize 40K products)

Predictable behavior
- bounds on worst-case performance (time and memory)

Implement a reliable storage mechanism on unreliable medium (flash)
- even in the event of an unexpected reboot

Hard deadlines!

Challenges

Processor speed 133 Mhz

Main memory 128 MByte (same as MER)

Over 65 Level-3 and Level-4 functional requirements

Support upto 300,000 data products
- fast retrieval of data in priority order for downlink
- support efficient catalog update (e.g., reprioritize 40K products)

Predictable behavior
- bounds on worst-case performance (time and memory)

Implement a reliable storage mechanism on unreliable medium (flash)
- even in the event of an unexpected reboot

Efficiently managing large sets with limited RAM

Different functions require efficient access via different orderings
- downlink → fetch oldest, highest-priority unsent product
- reprioritization → fetch all products in given time range
- autodeletion → fetch oldest, lowest-priority sent product

Well-researched problem in CS
- standard solution is to use *balanced search trees*

But…
- textbook red/black tree implementation requires 3 pointers/node
- → 3 * 4 Bytes * 300,000 product == 3.4 MB
  == ~ 2.7 % of RAM per tree

MSL uses a packed variant of Anderson trees (2 pointers/node)
- each tree requires ~ 1.4 MB of RAM

Challenges

Processor speed	133 Mhz
Main memory	128 MByte (same as MER)

Over 65 Level-3 and Level-4 functional requirements

Support upto 300,000 data products
- fast retrieval of data in priority order for downlink
- support efficient catalog update (e.g., reprioritize 40K products)

Predictable behavior
- bounds on worst-case performance (time and memory)

Implement a reliable storage mechanism on unreliable medium (flash)
- even in the event of an unexpected reboot

Decoupling access to shared memory

MSL FSW has ~ 140 tasks running concurrently
- some are running at 64 Hz / 8 Hz / 1 Hz
- others are background tasks

Any subset of them may want to write a data product at any time
- products are created in (fast) RAM filesystem
- later *migrated* to (slow) flash filesystem

Need to manage decoupled access such that
- a task can write to RAM filesystem *independently* of other tasks

MSL uses a RAM filesystem that provides
- *wait-free write and read access*

Challenges

Processor speed 133 Mhz

Main memory 128 MByte (same as MER)

Over 65 Level-3 and Level-4 functional requirements

Support upto 300,000 data products

- fast retrieval of data in priority order for downlink
- support efficient catalog update (e.g., reprioritize 40K products)

Predictable behavior

- bounds on worst-case performance (time and memory)

Implement a reliable storage mechanism on unreliable medium (flash)

- even in the event of an unexpected reboot

Why flash filesystems are hard

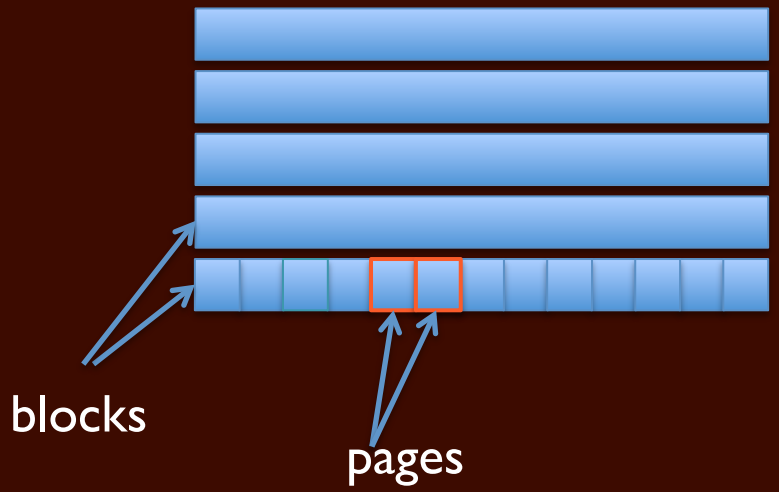
Asymmetry of write / erase

write_page
erase_block

Bad blocks

Limited block lifetimes

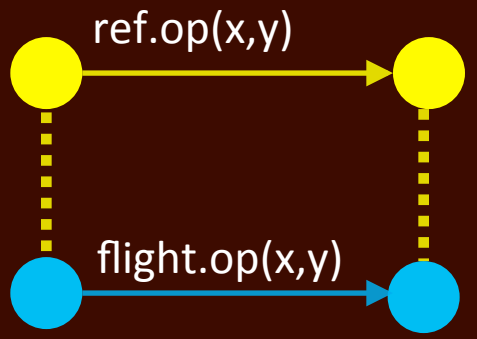
must do *wear-leveling*



Key property: "Reset-reliability"

filesystem operations should be atomic
even across an unexpected reset

Framework for filesystem testing




```
      choose op(x,y)
       [ inject fault in flight ]
      r0 := flight.op(x, y)
      r1 := ref.op(x, y)
(a)   assert   r0 == r1
(b)   assert   flight.state   ~   ref.state
(c)   assert   flight.invariants()
```

tree equivalence

data structure invariants; properties not modeled in ref

Golden Rule for testing

if either (a) or (b) fails

strengthen (c) until it also fails

Randomized testing

Simple to set up and maintain
- we used the model checker SPIN as a test driver
- instrumented code with CIL to measure *path coverage* to evaluate heuristics

Easy to parallelize

Surprisingly effective
- found many bugs that would not have been caught in system test

Several successes
- e.g., caching bug found after launch that could have resulted in all file creations failing silently after ~ 80 sols on surface.

One notable failure
- Sol-217 bug that caused safing
- Later reproduced by random tester (min scenario has 16 ops with 2 resets)

Randomized testing challenges

Overnight run returns many occurrences of the same bug

What to do when no more bugs are being found?

Less effective without an oracle