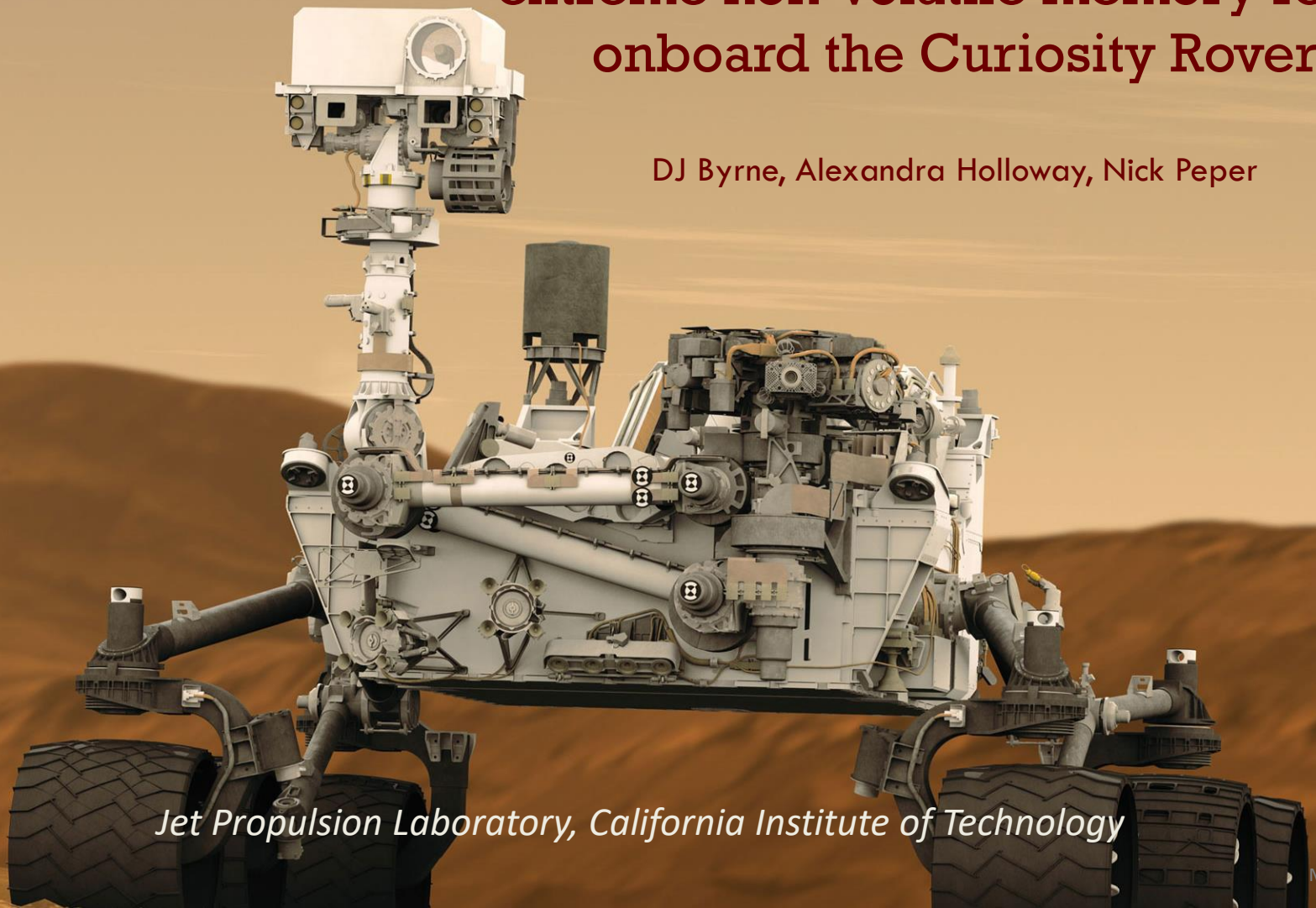
Mars Science Laboratory

R-Hope: Development approach to extreme non-volatile memory reuse onboard the Curiosity Rover

DJ Byrne, Alexandra Holloway, Nick Peper

Jet Propulsion Laboratory, California Institute of Technology



Agenda and Thanks

Agenda for this talk

- The Situation
- The Approach
- Approach Architecture
- Development Details
- Lessons Learned

Software development team

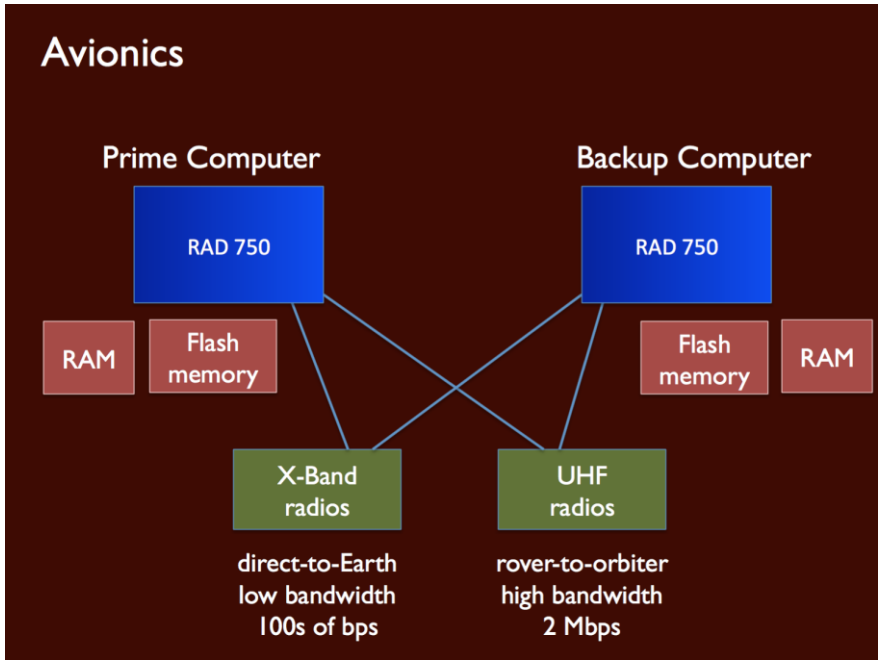
Ron Baalke
DJ Byrne
Matthew Dailis
Alexandra Holloway
Nick Peper

Added support

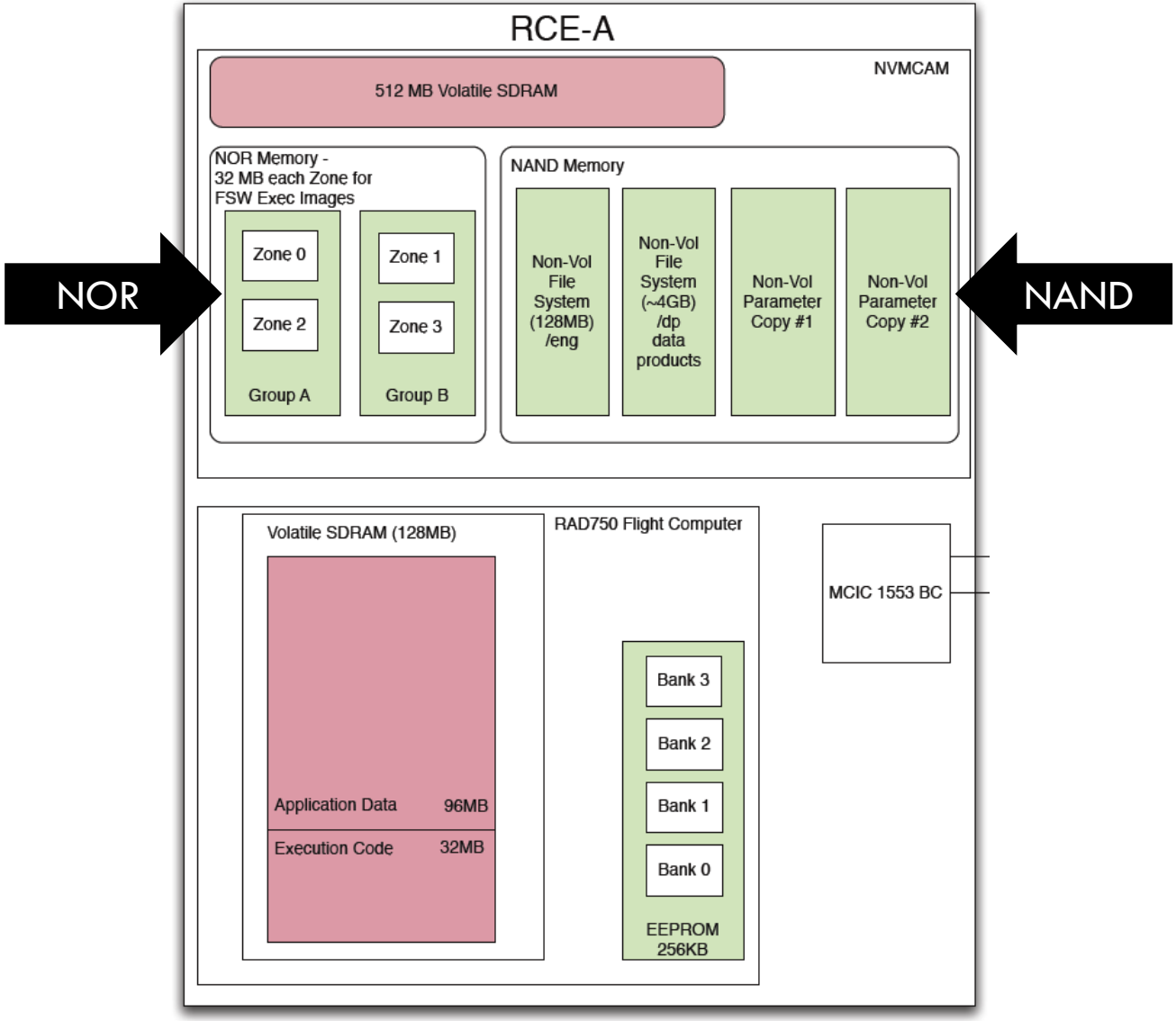
Magdy Bareh
Eddie Benowitz
Jim Butler
Tim Canham
Johnny Chang
Jonathan Denison
Jim Donaldson
Greg Dubois
Jim Erickson
Sierra Flynn
Dan Gaines
Rishab Gangopadhyay
Evan Graser
Jason Heidecker
Tom Huynh
Rajeev Joshi
Brian Kahovec
Kyle Kaplan
Mark Katsumura
Matt Keuneke
Gregory La Borde
Reidar Larsen
Daniel Limonadi
Megan Lin
Sean McGill
Camden Miller
Andy Mishkin
Alex Moncada
Jim Montgomery
Ryan Mukai
Deep Mukherji
Tracy Nielson
Marc Pack
Nik Patel
Neel Patel
Betina Pavri
Art Rankin
Kim Rink
Anna Sabel
Steve Schroeder
Garett Sohl
Rob Steele
Sloan Swieso
Matthew Van Kirk
Monica Wang
Jim Wang
Daniel Zayas
Jason Zheng

Introduction: Curiosity hardware/software setup

- Curiosity has been on the surface of Mars since August 5, 2012
  - How old is *your* laptop?
- Curiosity has 2 Rover Compute Elements called ***RCE-A*** and ***RCE-B***
  - Prime: Runs the mission
  - Backup: Only boots when Prime fails, then takes over the mission
    - Always high priority is fixing the RCE that failed
  - Identical hardware and software
  - Configuration files and parameters kept in sync



Approach Architecture: Physical (RCE-A)



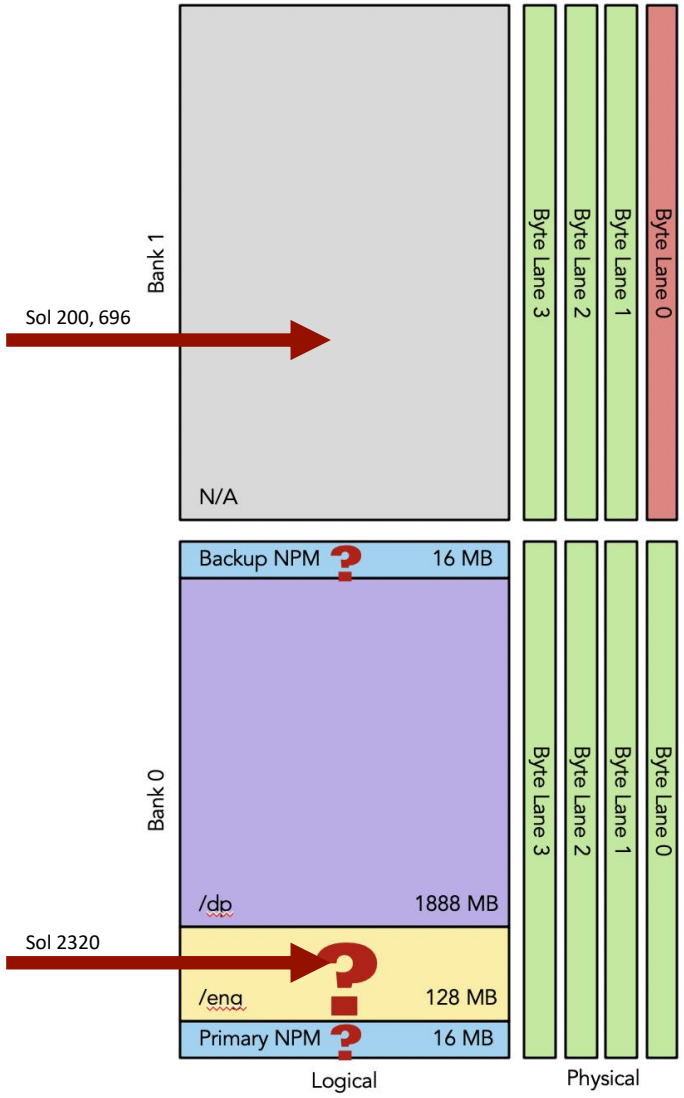
The Situation: Suspect memory

On Mars, a non-volatile memory chip holding the filesystem and parameters in RCE-A began failing.

- Sol 200: Partial failure. Resulted in re-mapping NV partitions to use only ½ of the chip to avoid the failed component
- Sol 696: Worsening failure. Resulted in mapping out all bad memory from RCE-A FSW
- Sol 2320: New failure presenting as intermittent failure of the entire chip

We swapped Prime and Backup, so RCE-B became Prime and mission continues

What will we do if RCE-B fails as well?



The Approach: A lifeboat for RCE-B

R-Hope will make RCE-A "safe for safing"

- Maintain control over thermal, power, communication, etc.
- Store recorded telemetry for later downlink
- Execute sequencing and fault protection

Re-establish filesystem and parameters... somewhere else

- But where

The failing chip is 4GB… where can we find so much space?

- We can't

https://www.uuworld.org/articles/lifeboat-faith

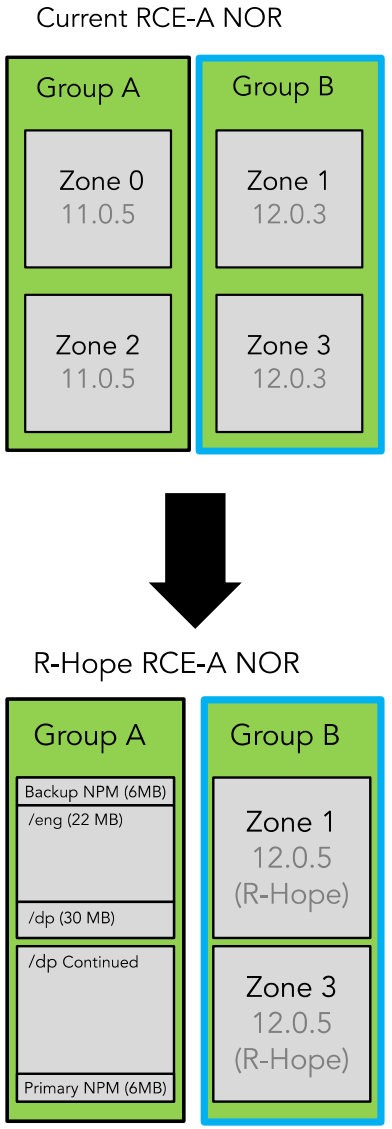
The Approach: A *tiny* lifeboat for RCE-B

Let's talk about NOR

- Four copies of the flight software image are stored on-board, each in a separate non-volatile memory "Zone"
- The four Zones are independent of the failing chip
- ...We don't really need four copies of FSW, do we?

Lean and Mean

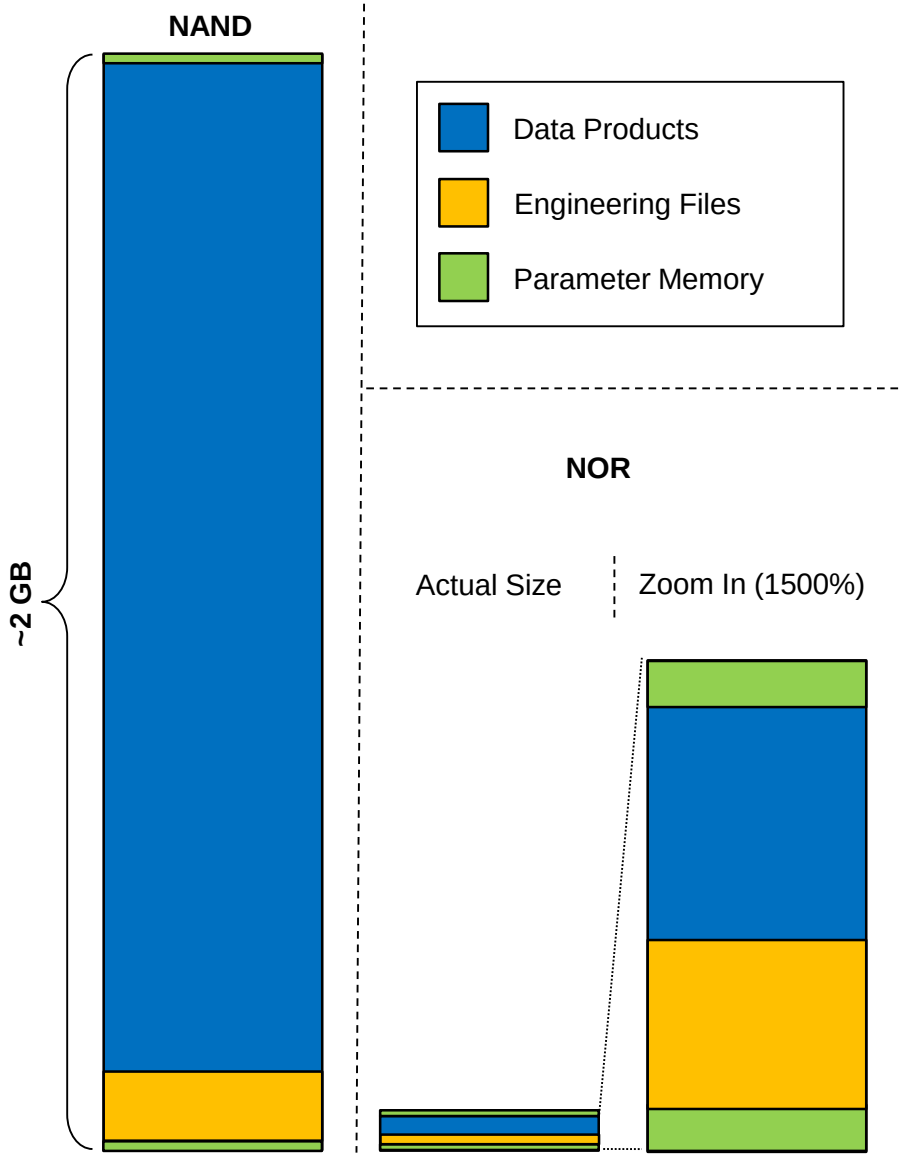
- NAND: The failing chip is 4 GB
- NOR: Each Zone is 32 MB
- If we re-purpose two Zones, is 64 MB enough to run a mission?
- Yup! ...Not gonna be saving videos, though



Approach Architecture: Physical

	NAND	NOR
Total size	4,192 MB	64 MB
Unit size	512 MB blocks	16 and 128 KB sectors
Bad units?	Bad blocks identified and mapped out	No known bad sectors
Erase cycle lifetimes	100,000	1,000,000
Repopulation time (format)	3.8 minutes	40 minutes
Access times	Read and write speeds comparable; erase ~10x slower	
Address translation efficiency	Naïve approach: 66% Robust approach: 98%	
Write Protection	None	Non-Volatile and Volatile Relays Restrict Write and Erase

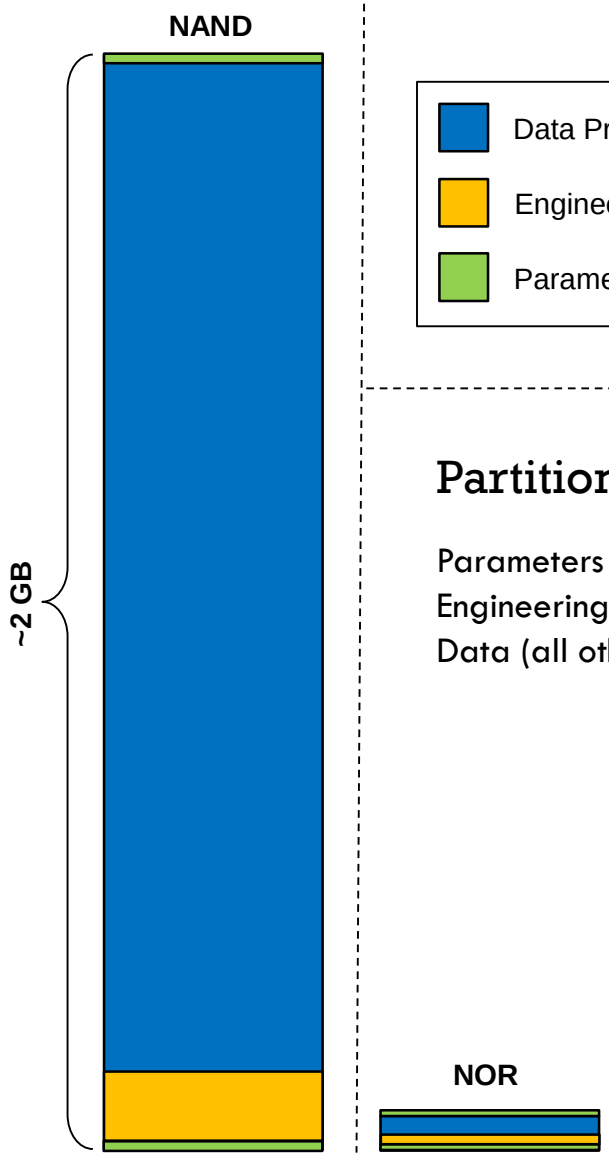
Significant Reduction in Size



Partition size reductions

Parameters (2 copies): 63%
Engineering: 83%
Data (all other): 98%

Significant Reduction in Size



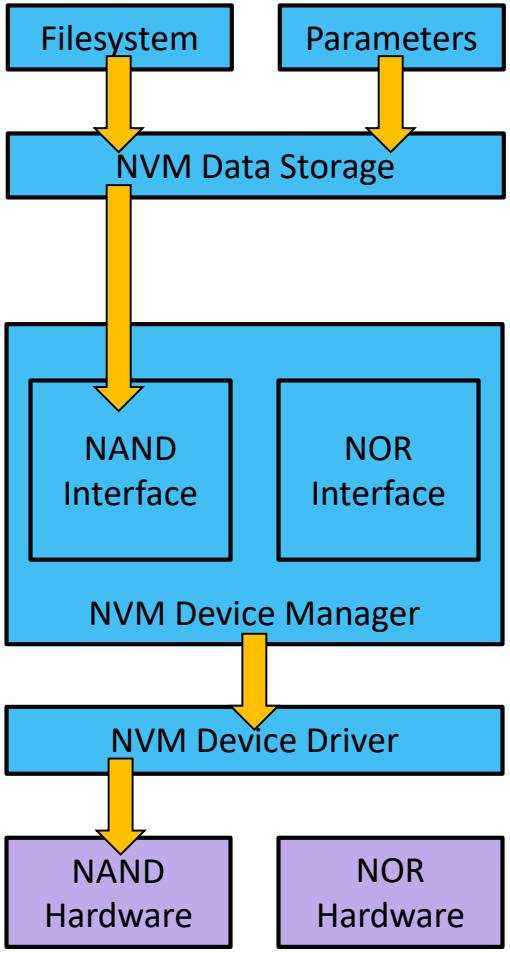
Data Products
Engineering Files
Parameter Memory

Partition size reductions

Parameters (2 copies): 63%
Engineering: 83%
Data (all other): 98%

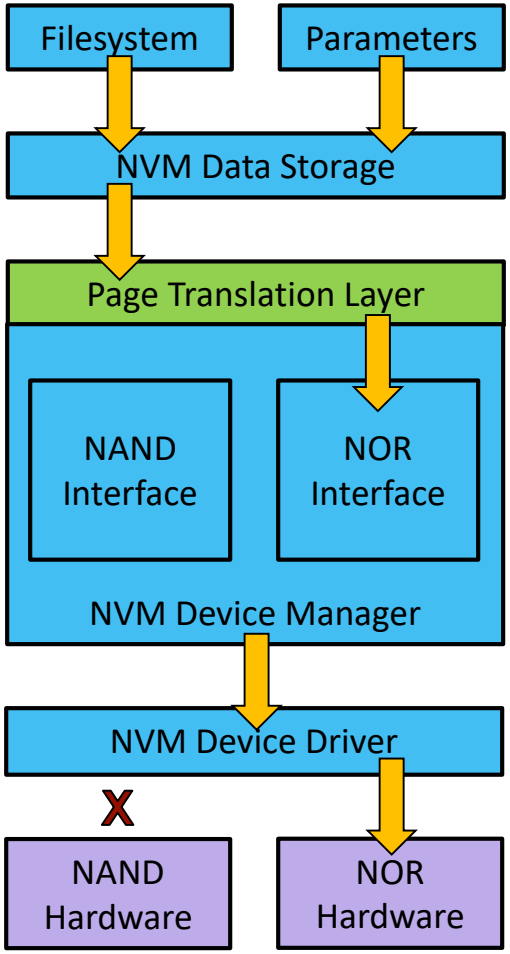
Approach Architecture: Logical

Before:



Unused by filesystem

R-Hope:



Development Approach Challenges (1/2)

1. Can we run a mission with this little data storage?
2. Will NOR chip performance be adequate?
   - NOR only read on average 5 times per sol; written once a year
   - NAND has copy-page in hardware; would have to do in software for NOR
3. Structural differences to account for in software
   - Differences between NAND and NOR
   - Existing code tries to abstract these differences away, but assumptions "leaked" upward, especially the concept of Page numbers
4. Testing limited mainly to hardware
   - Later development tested exclusively via unit test and hardware testbed
     - Only one hardware testbed named MAGGIE, a high-value asset
     - Tuesday/Thursday cadence
   - Why not a software simulator? Ours did not model the NOR chip at the time

Development Approach Challenges (2/2)

5. Options when choosing a code base
   - Existing R12, which has been running on Mars since 2014 and contains "Special code" for RCE-A, using only ½ of NAND chip
   - In-progress release R13, for flight "soon"
   - Code from SMAP (Soil Moisture Active/Passive), a non-Mars mission that inherited and continued development on the MSL code
   - Code from Mars 2020 Perseverance which inherited MSL code, made many improvements
   - Start from scratch. HAHAhahaha!
6. Difficulties with configuration management
   - Previous release of flight software (R12), this release (R-Hope), and next release (R13) all co-exist in a single repository using custom code over SVN
7. Brand new team with little exposure to existing software
   - And main filesystem expert retired the day before

DESIGN AND IMPLEMENTATION DETAILS

DJ Byrne

Code Base We Ultimately Chose

- **R12**, with as few changes as possible
  - Only code already proven on Mars-surface
  - Most familiar to operations team. If we're in the lifeboat, that counts for a lot
  - Have to rip out the "special version" patches
- Plus bug fixes cherry-picked as relevant to a lifeboat, from
  - MSL R13, SMAP, Mars 2020
- Eschew new functionality without removing code
  - E.g. arm, driving, instrument use, engineering cameras

How We Broke Up The Deliverable Into Stages

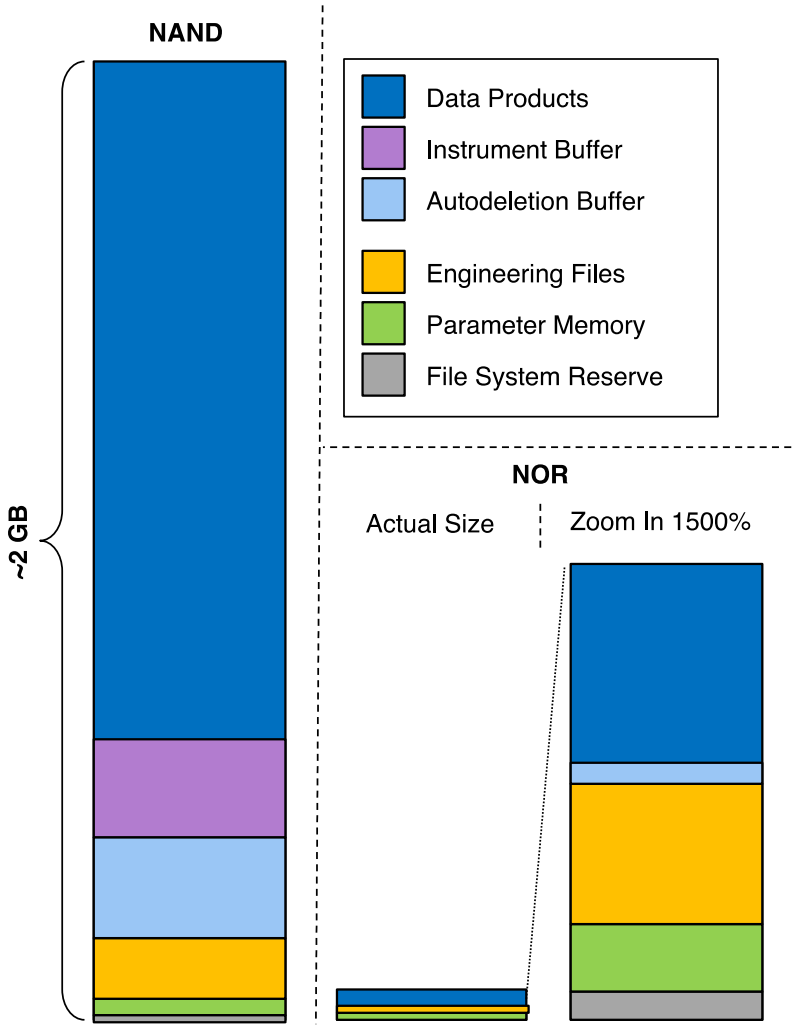
- Decompose task into major builds to retire risk in stages
- V0.x Unit Test fixes and baseline
  - Challenge: Configuration Management branched from R12
  - Challenge: Before changing code, updated existing unit tests and preserved baseline outputs
  - Advantage: All future code changes tested against this baseline
- V1.x Shrink NAND usage
  - Challenge: Can we run a mission with this little space?
  - Advantage: Can still use software simulator
- V2.x Use NOR for NAND
  - Challenge: Will NOR performance be adequate in new regime?
    - Mind the watchdogs
- V3.x (optional, then deferred) Tuning
  - Book-keep adjustments and optimizations that may turn up in testing
  - V2.x performance was actually good enough we stopped there
  - "Done is Better Than Perfect"

V0.x Unit Testing

- Most unit tests continued to pass despite release being 5 years old
- Some unit tests required modification
- Static analysis was performed to establish baseline for risk management
  - Coverity to find and fix software defaults and apply MSL-specific and Power-Of-Ten (P10) rules
  - Valgrind, a tool introduced late in MSL development to detect memory management issues such as uninitialized memory (and memory leaks)
  - gcov with unit test to gauge code coverage
- It was difficult, let's just leave it there

V1.x Shrink NAND Usage

- Capacity planning - Iterative Best Guess
  - Minimum parameter usage based on known and static R-Hope volume needs plus any future updates
    - Reclaim built-in spare capacity
    - Flight parameter capacity on R-Hope will allow at least 4 future FSW updates before running out of space
  - Minimum Engineering file storage space constraints driven by FSW image files
    - R-Hope must be capable of storing a full, compressed 10MB FSW image
  - Does that leave enough for data files?



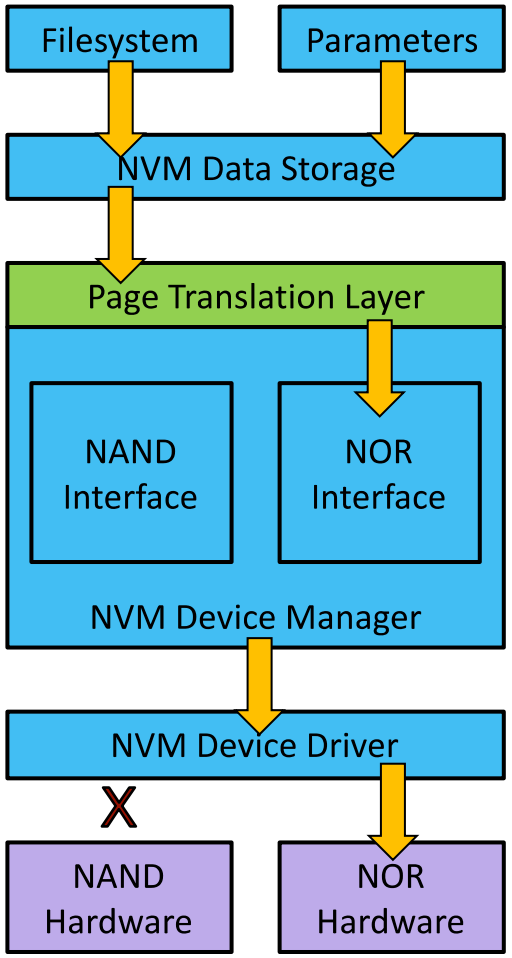
V1.x Shrink NAND Usage

- Can test these changes using software simulator - Yay!
  - Ironically, it only simulates 512MB of the 8GB NAND space for efficiency
  - So we updated it to use the shrunken size, and it's now more flight-like than it had been
- Patch in bug fixes from SMAP and Mars 2020 while we have simulator access

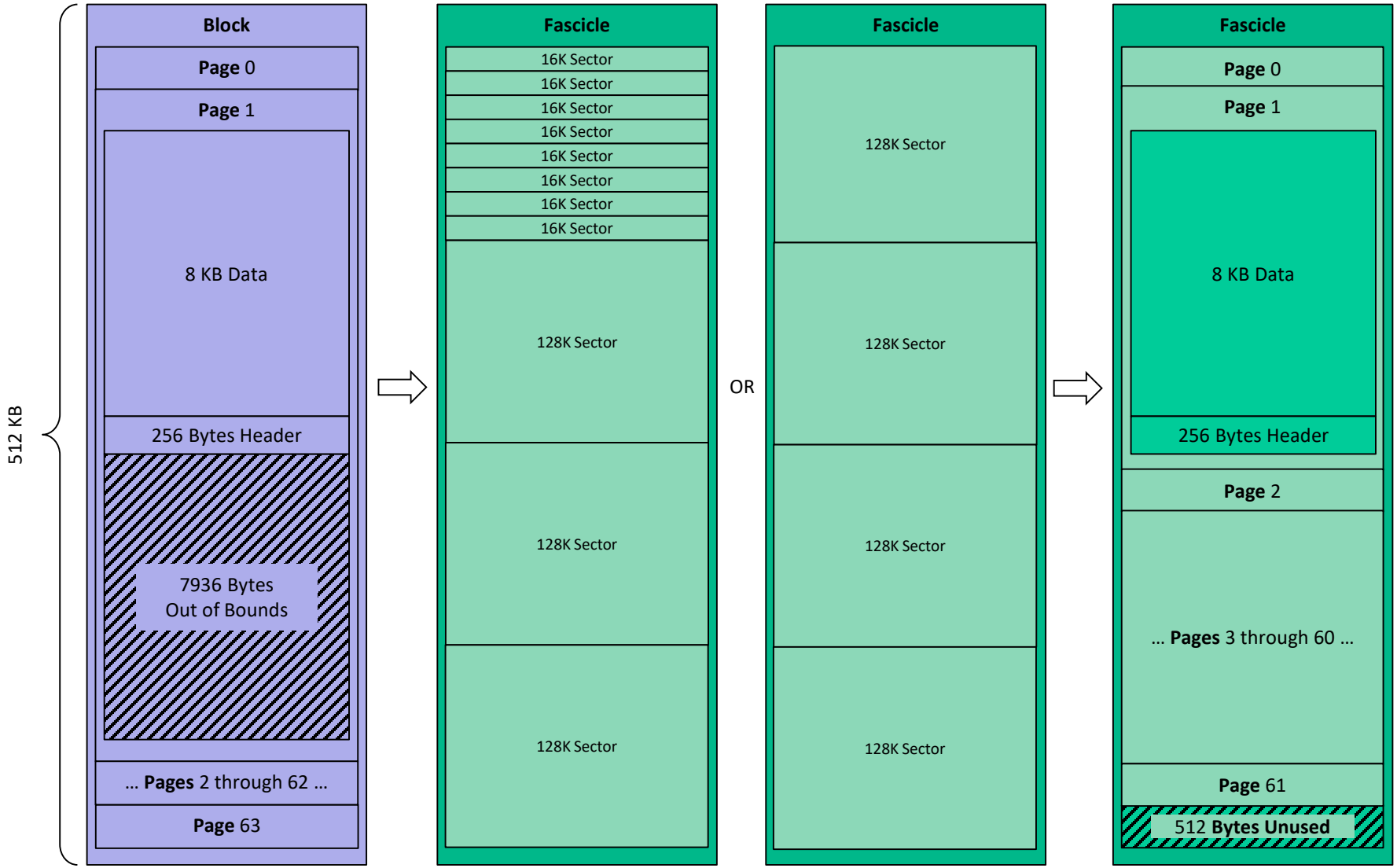
V2.x NOR-for-NAND: Translation

- Replace Data Storage calls that operate on NAND
  - With functionally equivalent calls operating on NOR
  - Read, Write, Erase, Copy-Page
- Map between NAND (block/page) addressing to NOR (zone/sector/offset)
  - Account for both data and header from NAND pages
  - Account for two different NOR sector sizes
- Copying
  - NAND has an interface to copy a page in hardware
  - NOR does not; have to create an interface in software

R-Hope:

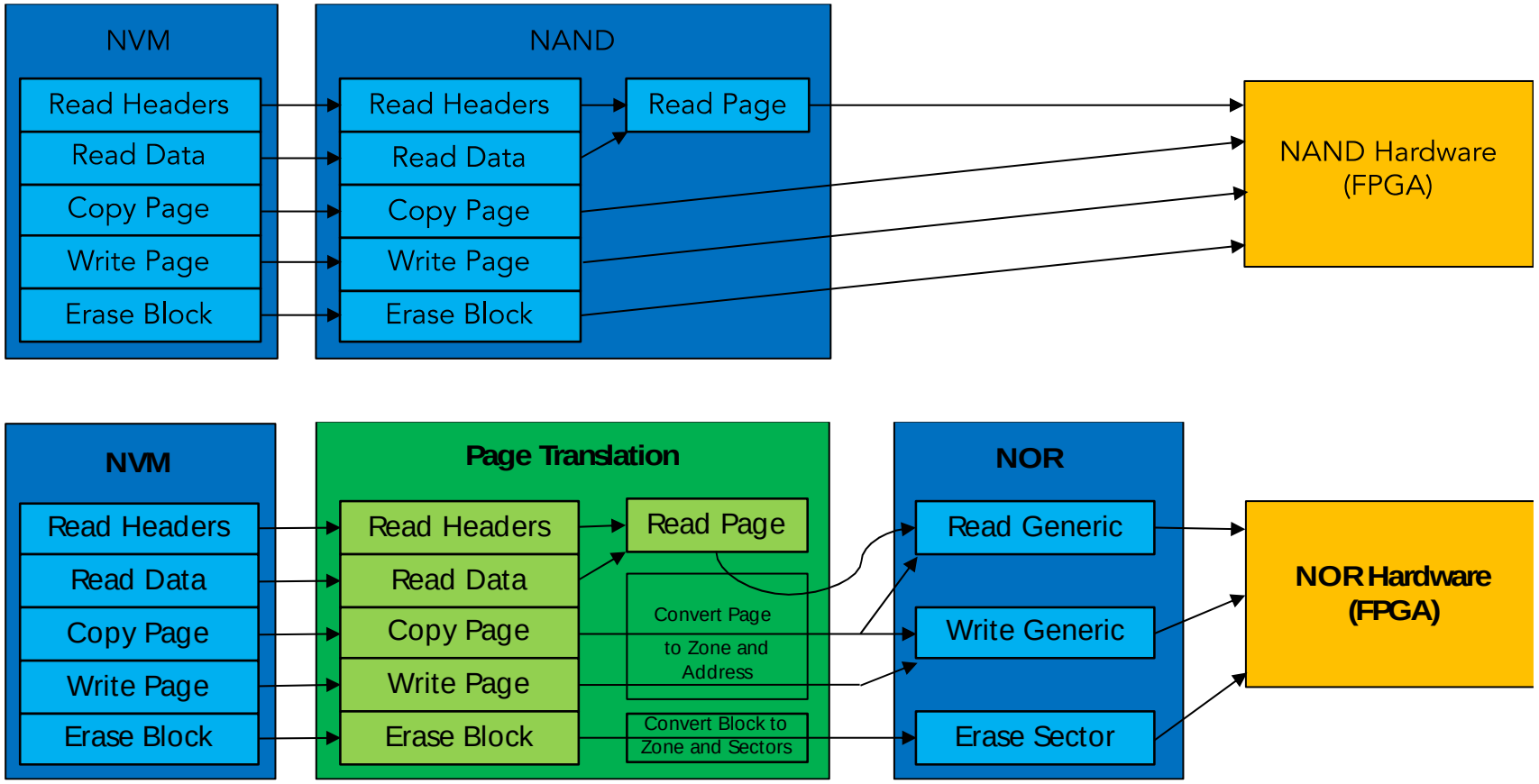


V2.x NOR-for-NAND: Translate Blocks to Fascicles



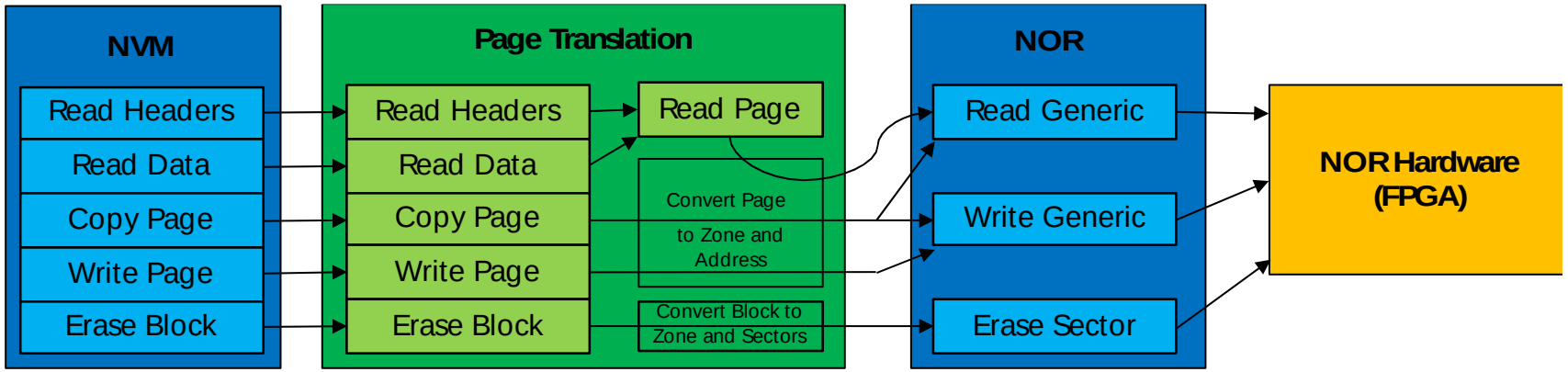
Note: Block size decreased from 64 pages on NAND to 62 on NOR

V2.x NOR-for-NAND: Page Translation Layer



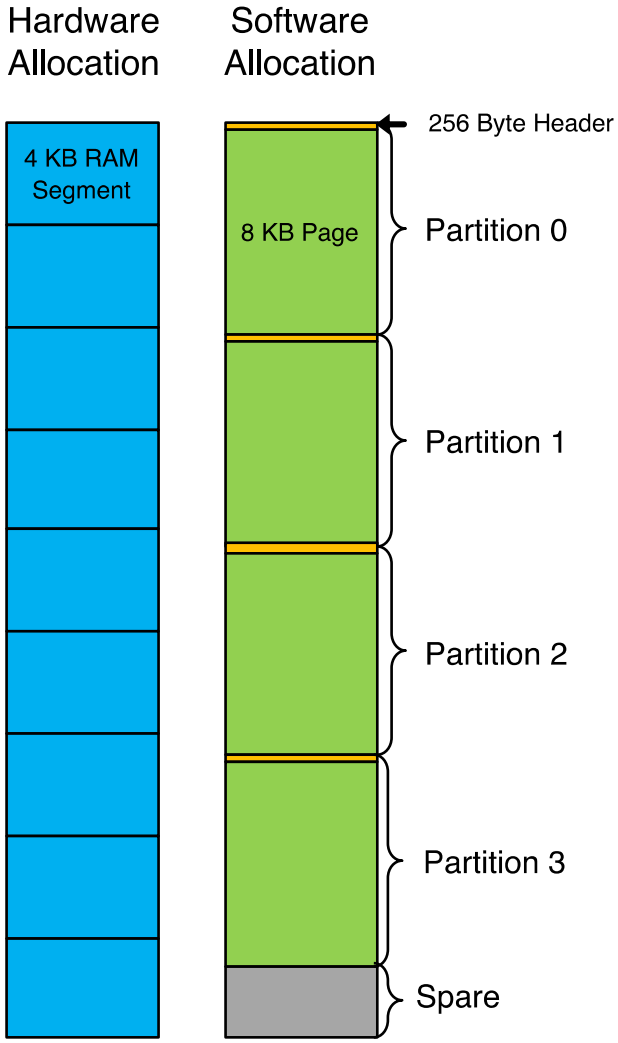
V2.x NOR-for-NAND: Page Translation Layer

- Read/Write
  - High-level code (above new translation layer) operates on page numbers
  - R-Hope translates a high-level operation on "Page number *aaa*" to "Zone *bbb*, Offset *ccc*"
- Copy Page
  - NAND does this in hardware, using page numbers
  - R-Hope does a combination read+write
    - Page headers are not copied; a new one is created
- Erase Block
  - Re-imagine NAND blocks as NOR "Fascicles" -- All the good names were taken: (block, page, zone, sector, segment, chunk, box...)
  - Conceptually, a Fascicle is "the smallest thing we can erase"
  - Underneath the translation layer, a Fascicle is a set of NOR sectors



V2.x NOR-for-NAND: DMA (Direct Memory Addressing) Segment

- Read and Write operations to the NOR chip are done via FPGA with DMA
- DMA operations must be on contiguous memory segments
  - Our system uses 4KB segments
  - Sadly, NAND operations do not require that Page Data and Header are contiguous
- R-Hope adds a DMA-safe scratch pad
  - How big a scratch pad do we need?
    - Big enough for four pages
      - High-level code ensures only one Page operation per each of the four partitions at a time
    - Each Page is 8KB Data + 256 Bytes Header, or just over 2 segments
    - Nine memory segments will hold four Pages
  - Carved it away from other modules
    - Luckily, the allocation at the end of memory was a good choice
      - Didn't shift the memory map for anything else
      - Easier to review the change



V3.x (Optional) Tuning

- V3.x was planned but not needed
  - Avoided pre-mature optimization
  - Book-kept ideas for things to tweak later if needed
  - Who knows? Maybe three years from now Curiosity will really need another 3MB of storage space (ha!)
- Examples (including scary heritage breakers)
  - Page Headers could be shrunk to actual usage
    - Perhaps saving 128 Bytes per Page, or ~1MB total
  - Coarse granularity of Pages (8KB) might be wasting space
    - NAND Pages are this size by definition, and we chose not to change it
  - Strip out unused modules, if R-Hope will never do some activities
    - Reduces need for Engineering filesystem capacity
  - Hard-code selected parameters we don't need to change

HOW DID WE KNOW V2 WAS GOOD ENOUGH?

Nick Peper

Testing R-Hope

- By necessity with our constrained timeline, most R-Hope test procedure development, and some testing, was conducted in conjunction with code development
  - Testbed verification of desired functionality
- Once a final build of R-Hope was produced, final coordinated test and follow-on regression campaigns were performed on hardware testbeds
  - Test procedures were designed to flex new code as well as verify that old code relying directly on the new architecture was performing per expectations
  - Old regression procedures were performed as well for all FSW functionality required to achieve R-Hope's primary objectives
    - All capabilities required to be considered "safe for safing"
      - Fault protection, telecom, power, thermal, etc.
    - Cross-string diagnostic capabilities for recovering a failed RCE-B
  - This campaign was performed over 1 month with over 15 supporting testbed, systems, and subsystem engineers

Ops considerations

- Operations on R-Hope are constrained to maintenance of overall rover health and diagnostics on RCE-B
  - Planning rules and conventions constrain nominal operations of R-Hope
  - Inherited capabilities can be tested on the ground and approved for flight as needed
- Functionality recovery efforts involving R-Hope will present challenges related to hard limits on data collection
  - About 15MB data can be collected between downlinks before onboard data loss becomes a concern
- Assured presence of working backup flight computer reduces mission risk significantly
- R-Hope is currently running on MSL's RCE-A with checkouts completed on all its new memory partitions
- Operations on R-Hope is exactly the same as previous (R12) and future (R13) releases, reducing ops risk significantly

Key Dates

Milestone	Date
Main filesystem expert retires	2019-02-15
Initial presentation of memory issues	2019-02-15 (Sol 2320)
First presentation of R-Hope concept	2019-04-25
Original software request approved	2019-05-28
Development begins	2019-06-24
Development complete	2020-01-07
Unit tests and simulation released	2020-03-09
Testing complete	2020-04-13
Final approval for installation	2020-09-04
Uplink campaign ends	2020-11-02
Final approval by change control board	2020-11-29
Installation on Curiosity	2020-12-07 (Sol 2963)

This release represents 19 months of work by over 70 people – fast and furious output!

Lessons Encountered

- The MSL software proved itself modular and adaptable
  - Able to change as little flight code as possible; did not remove capabilities
- Agile-inspired process allowed a small team to work efficiently pivot easily due to new information and constraints
  - Parallel development for iterative dev-testing
  - Able to address new challenges related to hardware/software interface constraints previously unknown to the team
- Development would benefit from cross-mission communication on bug fixes
  - Informal communication channels (e.g., coffee, hallway chats) and co-located teams contribute to faster understanding
- Establishing unit test and static analysis baseline in heavy-heritage software projects is very important to understanding introduced risk
- Shrink NAND first, cut-over to NOR as late as possible worked great
  - So we can use software simulator for the NAND portion of work
  - Divide challenges across releases
- Our tight focus on lifeboat-related bug fixes identified patches to pull forward into R13 from SMAP and Mars 2020