THE UNIVERSITY OF TEXAS AT AUSTIN

Toward Exascale Resilience

Part 5:

Processors and networks

Mattan Erez

The University of Texas at Austin

July 2015

Processors are expensive

– Redundant processors vs. redundant memory chips

Expectation of high reliability

What's in a processor?

- Lots of SRAM
  - Large SRAM arrays in caches
  - Smaller arrays in local caches, TLBs, predictors, …
- Lots of smaller latch-arrays
  - Buffers, registers, …
- Datapaths
  - Logic for doing compute – processing the data
  - **Pipelining means a lot of scattered latches**
- Control logic
  - **Pipelining and FSMs mean a lot of scattered latches**
- Communication
  - Buses, interconnection networks, ...

SRAM reliability dominates processor reliability

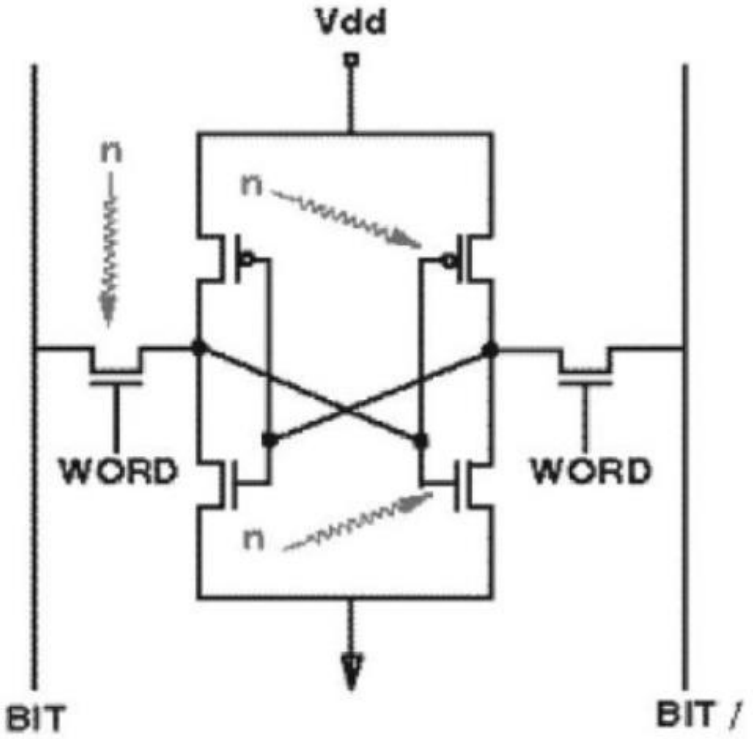
- Much more SRAM than anything else
- SRAM more vulnerable
  - Smaller transistors
  - Much less masking in memories than logic

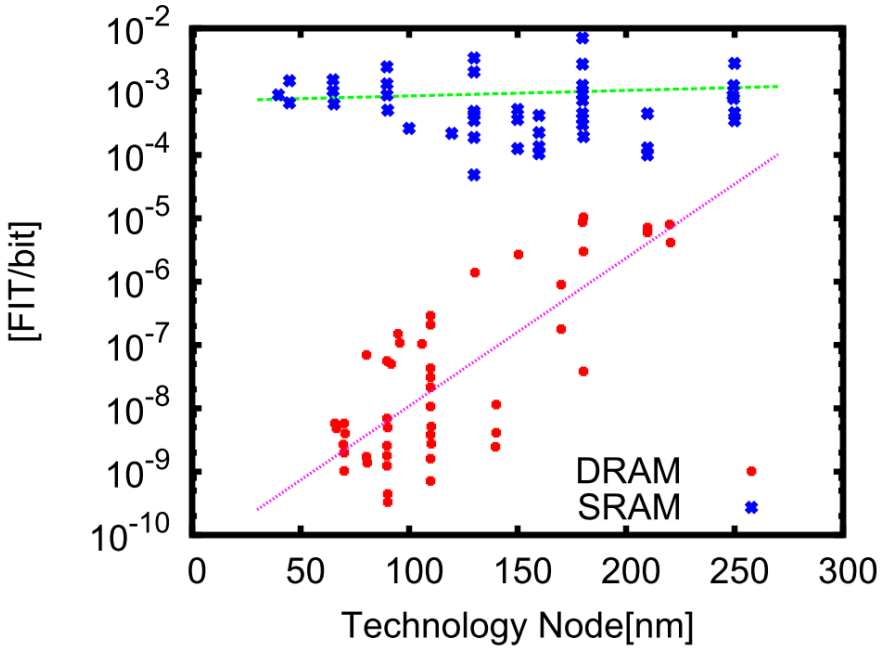
SRAM faults and error

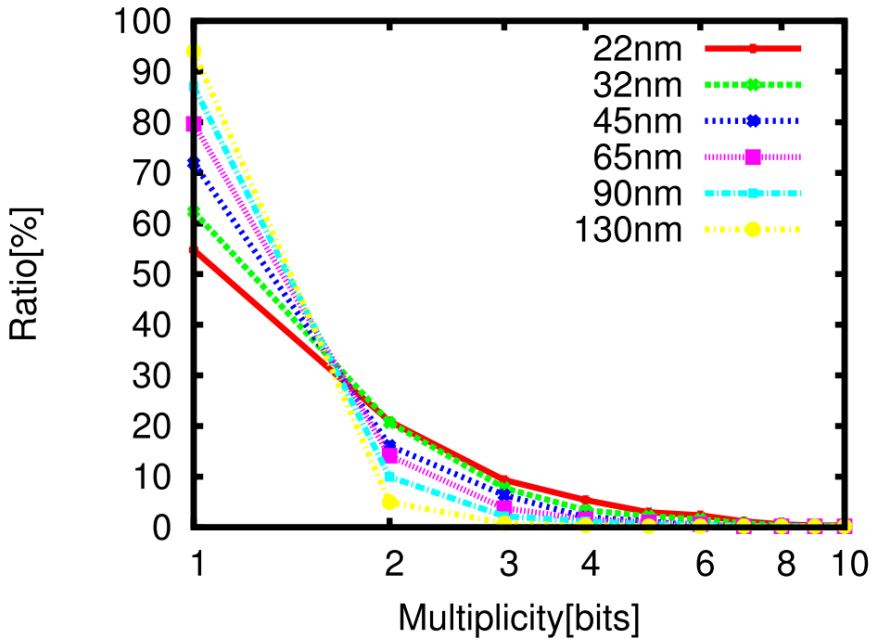
- Particle-strikes
- Retention errors
- Read and write errors

Particle strike faults

- Ionizing particle leads to spurious current flowing
- Can overcome feedback

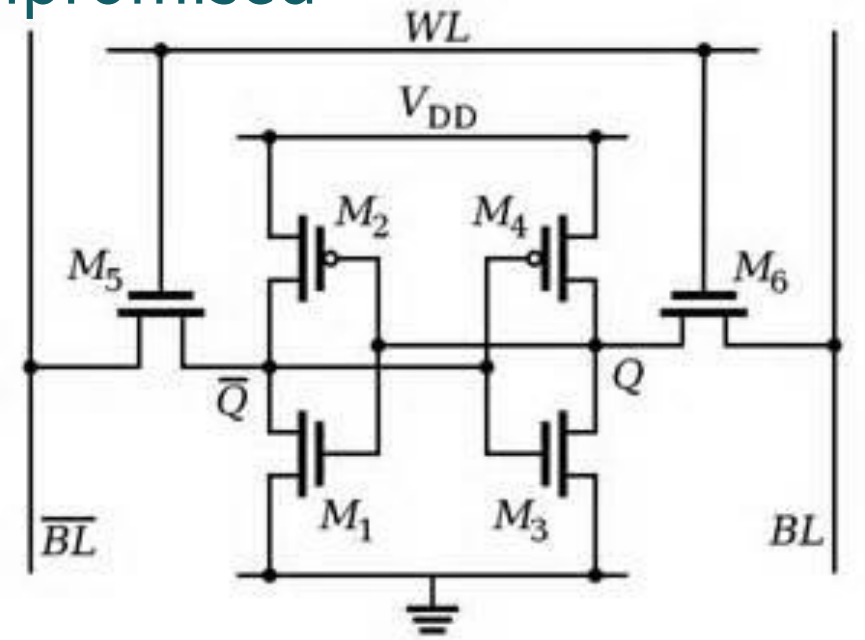


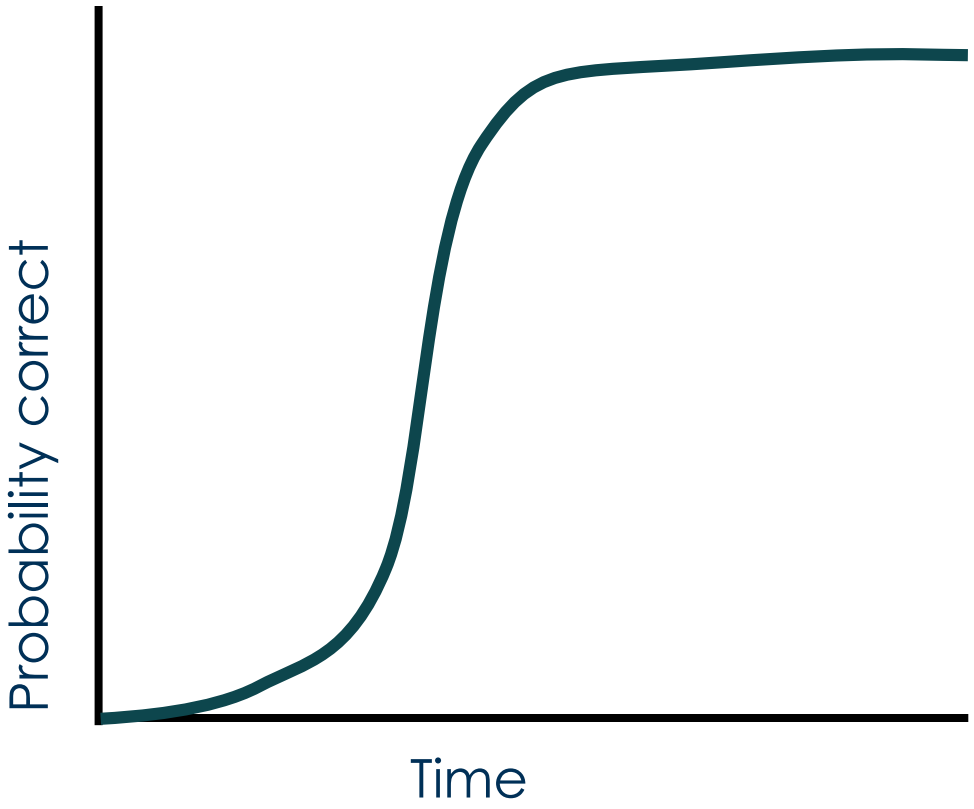




Retention and read/write errors

- Transistor mismatches lead to imbalance in symmetric SRAM cell
  - **Worse as transistors shrink because of variation**
  - **Worse as voltage decreases**
  - **Vcc_crit limits min voltage**
- Stability of feedback compromised
- Writes have longer tails
- Some reads too slow




Probability correct
Time

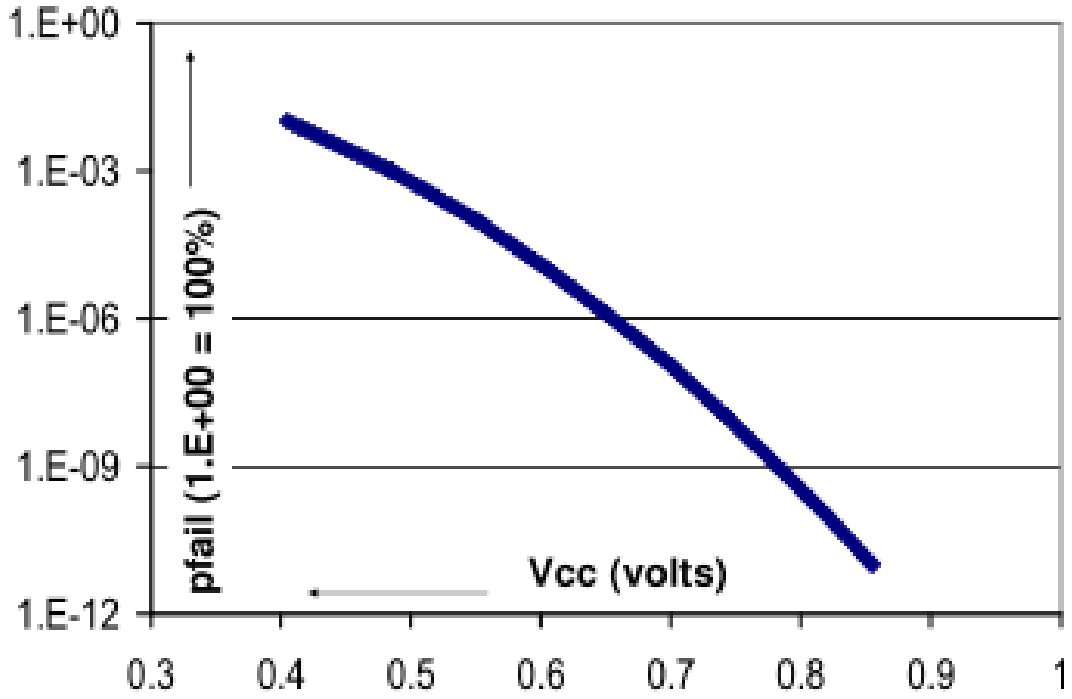


Figure 2. Probability of failure (Pfail) for a cell vs. Vcc [8]

From Wilkerson et al., ISCA'08

Circuit techniques

- Make cells bigger?
  - Use more transistors or bigger ones
  - Reduces error rates
  - Sacrifices too much area
- Clever read/write circuits
  - Help, but not with stability

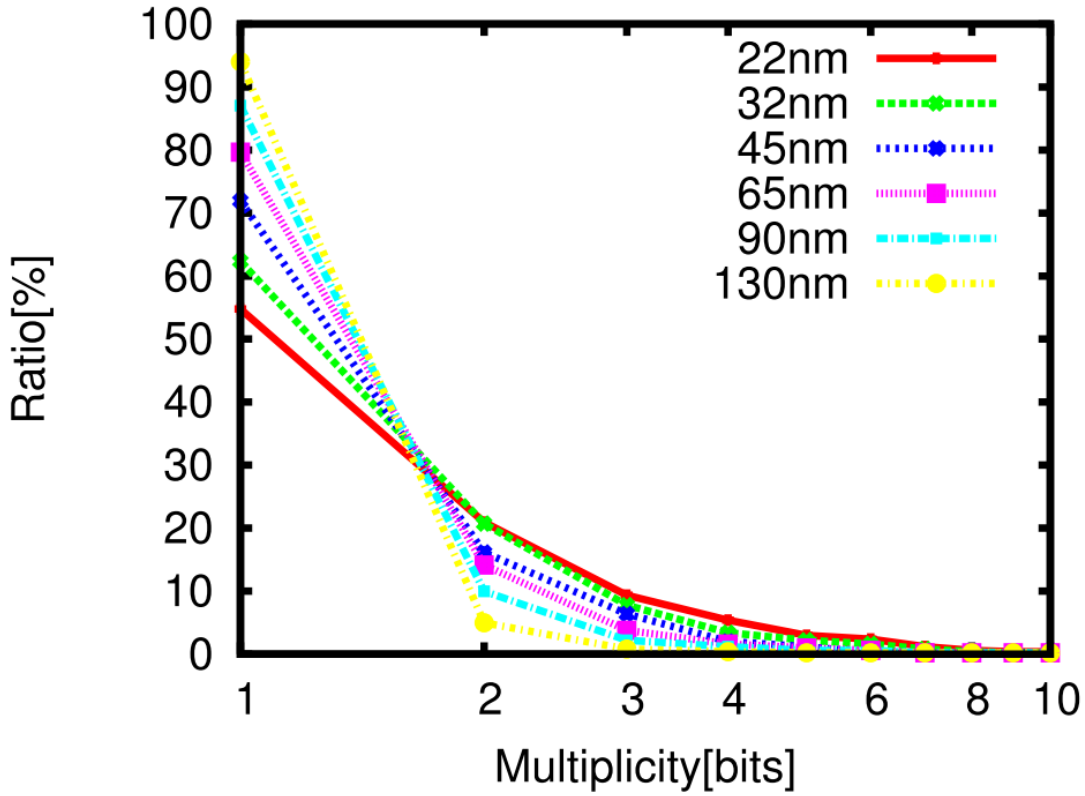
It's memory – use ECC!

- ECC has low redundancy
  - SECDED is 10 bits for 512b cache-line
  - DECTED just 17 bits
- Some ECC is simple to compute
  - Bit-wise hamming codes, in particular

How much ECC protection do we need?

How many bits can be wrong?

- Multiple particle-strikes very unlikely
- Variations very random
- But, multi-cell upsets exist



Correlations reduce protections costs

– Physical bit interleaving for large arrays (on board)

The try-try-again approach

- Detection → retry
  - Works great for transient errors that did not affect previous state

Backward (rollback) recovery!

Data in many arrays is replicated

- TLBs
- Write-through L1
- Clean cache lines, in general

Data in some arrays affects only microarch

- Predictors

Is detection enough?

- Yes! Can re-fill from elsewhere or regenerate
- Simple parity often enough

What about tags?

Remember the bear chasing you

- Balance protection with expected rates and impact
- Smaller structures often protected less
- No need to catch extremely rare events

Latch arrays

- Similar to SRAM arrays
- Bigger transistors → lower inherent fault rate
- Too many latches to ignore (in the future)

Use better circuits – **hardened latches**

- Design latches that check and correct themselves
- Basically replicate the latch
- Not cheap
  - More area and power, but manageable
- Can reduce by 10x or so

- Enough for exascale?
  - Not yet clear

ECC for latches?

- ECC is great, but need arrays to make effective
- Parity may be good enough for detection
  - Lower fault rate
  - More distance between cells (if not in array)
- Parity still sometimes hard to apply

Datapath

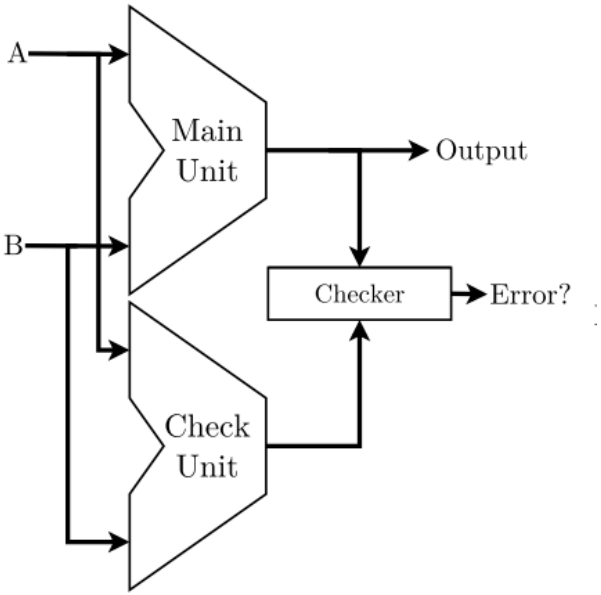
- Combination of logic and latches
- Used for numerical/logic operations
- Lots of masking
  - Often not a concern today
  - More expensive processors take care of it

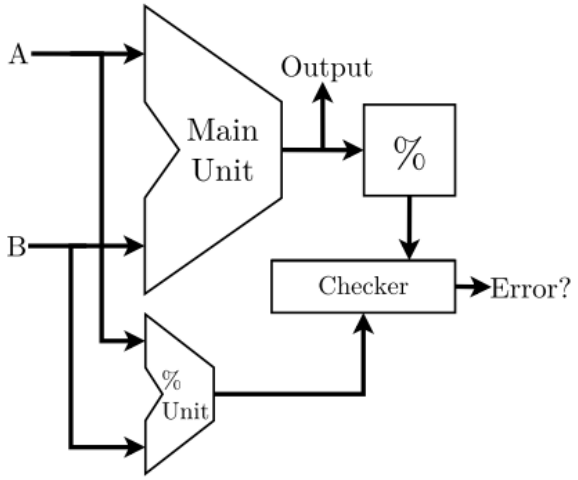
Protection through duplication

- Do everything twice
- In space?
  - Double the cost
- In time?
  - Double energy and time (of arithmetic)

Protection through reduced duplication

- Do we need to check everything to detect errors?
- Residue checking





$$|a \oplus b|_A \stackrel{?}{=} ||a|_A \oplus |b|_A|_A$$

Residue can be extended to protect registers

- Can this also catch other errors?

Protecting control

- Protect the logic
- Protect the semnatics

Protecting logic

- Without clear arithmetic, rely on design
- Harden the circuits
- Partial duplication → parity prediction
- >20% overhead
- Coverage hard to estimate, but assume good
  - Used in extreme designs

Protecting the semantics

- Check for symptoms of errors
  - Branching to an address that doesn't start a basic block
  - Illegal instructions
  - Out-of-bounds accesses
  - Using registers that haven't been defined
  - …
- Collection of symptoms may have excellent coverage
  - Evaluation tricky
- More on the board

Communication

- Buses move data between components
  - No changes to the data
- Parity (or other error detection) + retry
- More when we talk about networks
- Not problematic today
  - Because parity/retry is cheap and effective

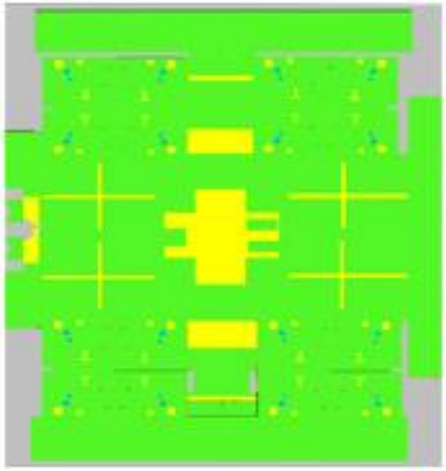
Conclusion: cost / reliability tradeoff

- We can build reliable processors for exascale
- Not clear that we should (more later)
- Processor count still a concern

From: http://www.fujitsu.com/global/products/computing/servers/unix/sparc/technology/reliability/processor.html

Fujitsu SPARC64 X

- Arrays + arithmetic + some control
- Not cheap
- My guess:
  - ~25% overhead in non-arrays



Circuits in Green : one bit error correctable
Circuits in yellow : one bit error detectable
Circuits in gray : no problem to continue operations even if this circuit fail

Table: SPARC64 X RAS Functions

	Error detection	Error Correction		Recording
		Correction	Degradation	
Level1 cache	Multiplicity Parity+ECC	Retry, ECC	Dynamic way degradation(*2)	Event recoding
Level2 cache	ECC	ECC	Dynamic way degradation(*2)	
Arithmetic Logic Unit	Parity (*1) +Residue	ECC, Hardware instruction retry	Core degradation	

Network

- Generally, multi-hop networks
- Source → NIC → routers/switches → NIC → dest

What's in a network?

- "Processor" + memory + links
- Backward recovery (retry) promenant

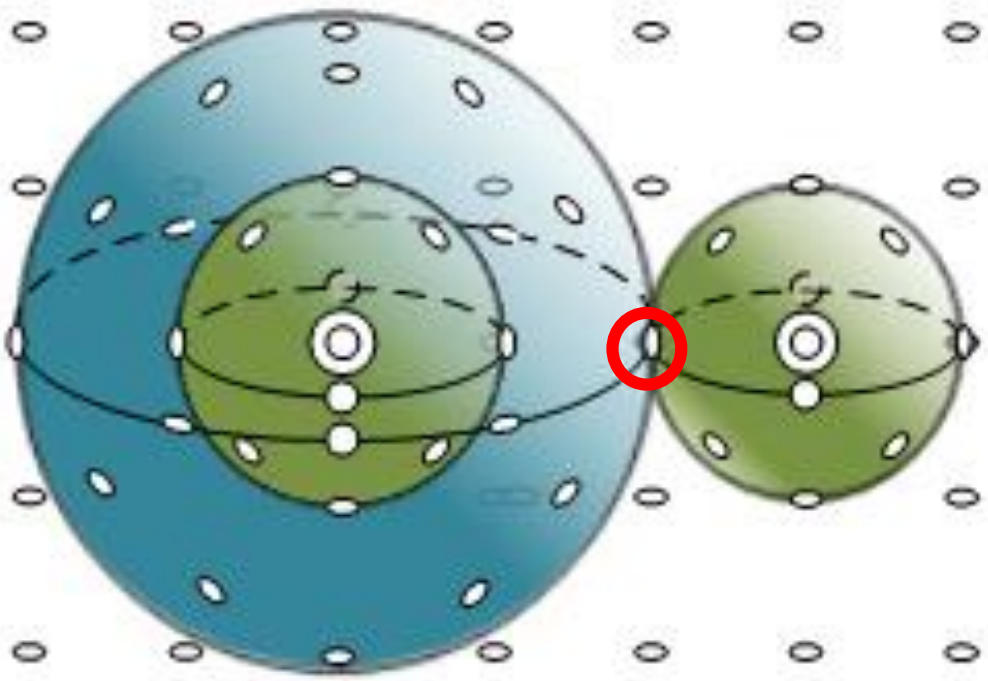
Network router (processor)

- Decides on routing
- Kind of looks like a simple processor
- Small part of overall network and typically well-protected with duplication and symptom checks

Memory is memory → use ECC

Links are like buses, carry the data

- Use error detection and retry
- Typically strong CRC for detection
  - Long-symbol codes that don't attempt to correct



End-to-end checks?

- Can protect the message too

What about failed links?

- Unfortunately, common
- Mechanical failures of connectors ☺

Path diversity and rerouting until repair!

Other hard faults

- Corrosion
- Mechanical stress
- Accidents
- Current stress

Mechanical → power → connectors

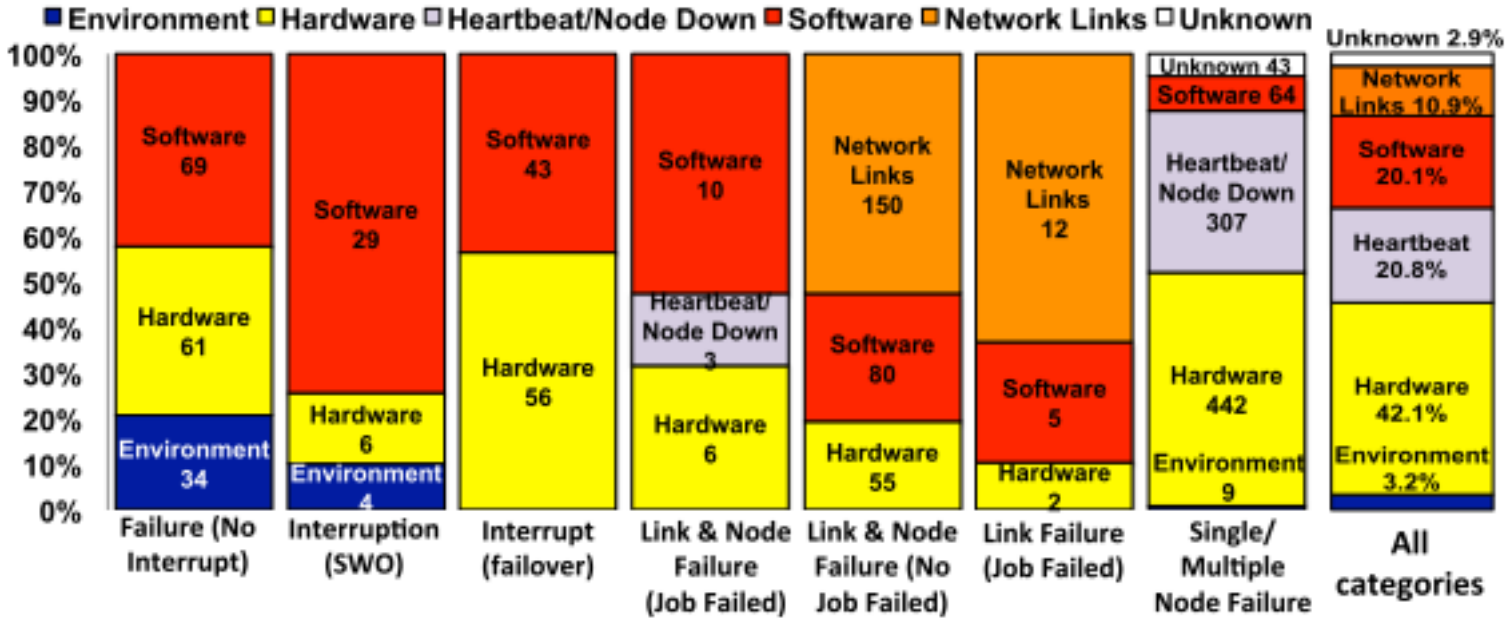
- Lots of redundancy can be put in
- Recently, scaling in the chip faster than outside

An example: Blue Waters

From Di Marino et al., DSN'14

TABLE III: Failure Statistics. The last row refers to the statistics calculated across all the failure categories.

Failure Category	count	%	MTBF [h]	MTTR [h]	σ_{TBF} [h]	σ_{TTR} [h]
1) Failure (No Interrupt)	164	11%	35.17	13.5	70.8	35.3
2) Interrupt (Failover)	99	6.6%	58	14.7	92	42.2
3) Link & Node Failure (Job Failed)	19	1.3%	297.7	6.1	427.3	5.4
4) Link Failure (No Job Failed)	285	19.1%	19.9	32.7	51.9	91.2
5) Link Failure (Job Failed)	19	1.3%	291.6	16	444	26.7
6) Single/Multiple Node Failure	868	58.2%	6.7	26.7	6.3	72
7) Interruption (system-wide outage)	**39**	**2.62%**	**159.2**	**5.16**	**174.2**	**8.1**
ALL	**1490**	100%	4.2	34.5	13.3	50.5



An example: Blue Waters

From Di Marino et al., DSN'14

TABLE III: Failure Statistics. The last row refers to the statistics calculated across all the failure categories.

Failure Category	**count**	**%**	**MTBF [h]**	**MTTR [h]**	σ_{TBF} **[h]**	σ_{TTR} **[h]**
1) Failure (No Interrupt)	164	11%	35.17	13.5	70.8	35.3
2) Interrupt (Failover)	99	6.6%	58	14.7	92	42.2
3) Link & Node Failure (Job Failed)	19	1.3%	297.7	6.1	427.3	5.4
4) Link Failure (No Job Failed)	285	19.1%	19.9	32.7	51.9	91.2
5) Link Failure (Job Failed)	19	1.3%	291.6	16	444	26.7
6) Single/Multiple Node Failure	868	58.2%	6.7	26.7	6.3	72
7) Interruption (system-wide outage)	**39**	**2.62%**	**159.2**	**5.16**	**174.2**	**8.1**
ALL	**1490**	100%	4.2	34.5	13.3	50.5

An example: Blue Waters

From Di Marino et al., DSN'14

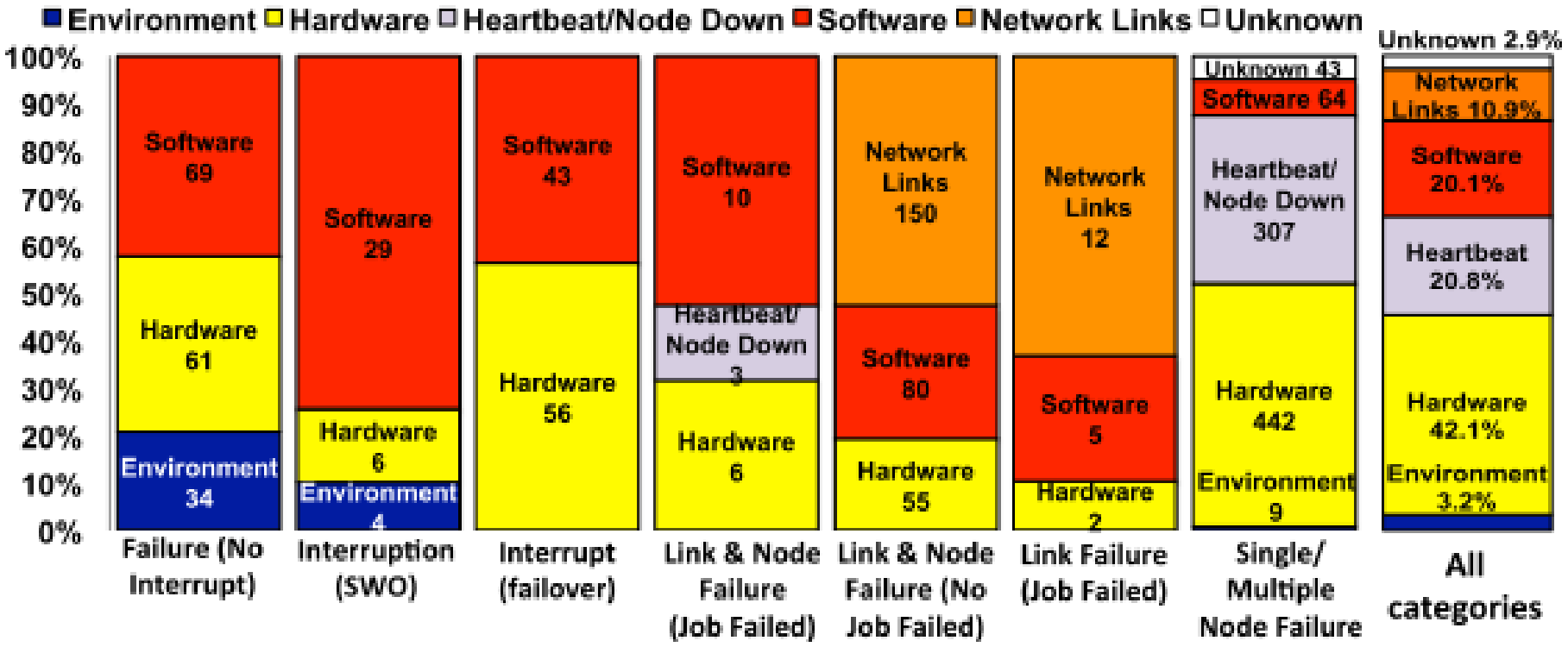


TABLE IV: Breakdown of the count of th

	Failure (No Interrupt)		Interrupt (SWO)		Interrupt (Failover)		I F
HW	PSU	20	EPO	1	Disks	45	
	IPMI	15	Compute Blade	2	IPMI	5	
	Fan tray assy	14	Storage module	2	Storage module	2	C
SW	Moab/TORQUE	33	Lustre	18	Lustre	29	
	CLE/kernel	17	Moab/TORQUE	6	Sonexion/storage	8	
	Warm swap	5	Gemini	3	CLE/	4	

From Di Marino et al., DSN'14

he top 3 hardware and software failure root causes

Link Failure (User Job Failed)		Link & Node Failure (User Job Failed)		Single/Multiple Node Failure	
Optic	12	GPU	2	Processor	160
RAM	9	Gemini ASIC	1	RAM	158
Gemini voltage regulator	8	Compute blade	2	GPU	38
Lustre net (Lnet)	2	Lustre	8	Lustre	30
		CLE/kernel	1	CLE/Kernel	16
				Sonexion/Storage	5