Self-Repair for Robust System Design

Yanjing Li

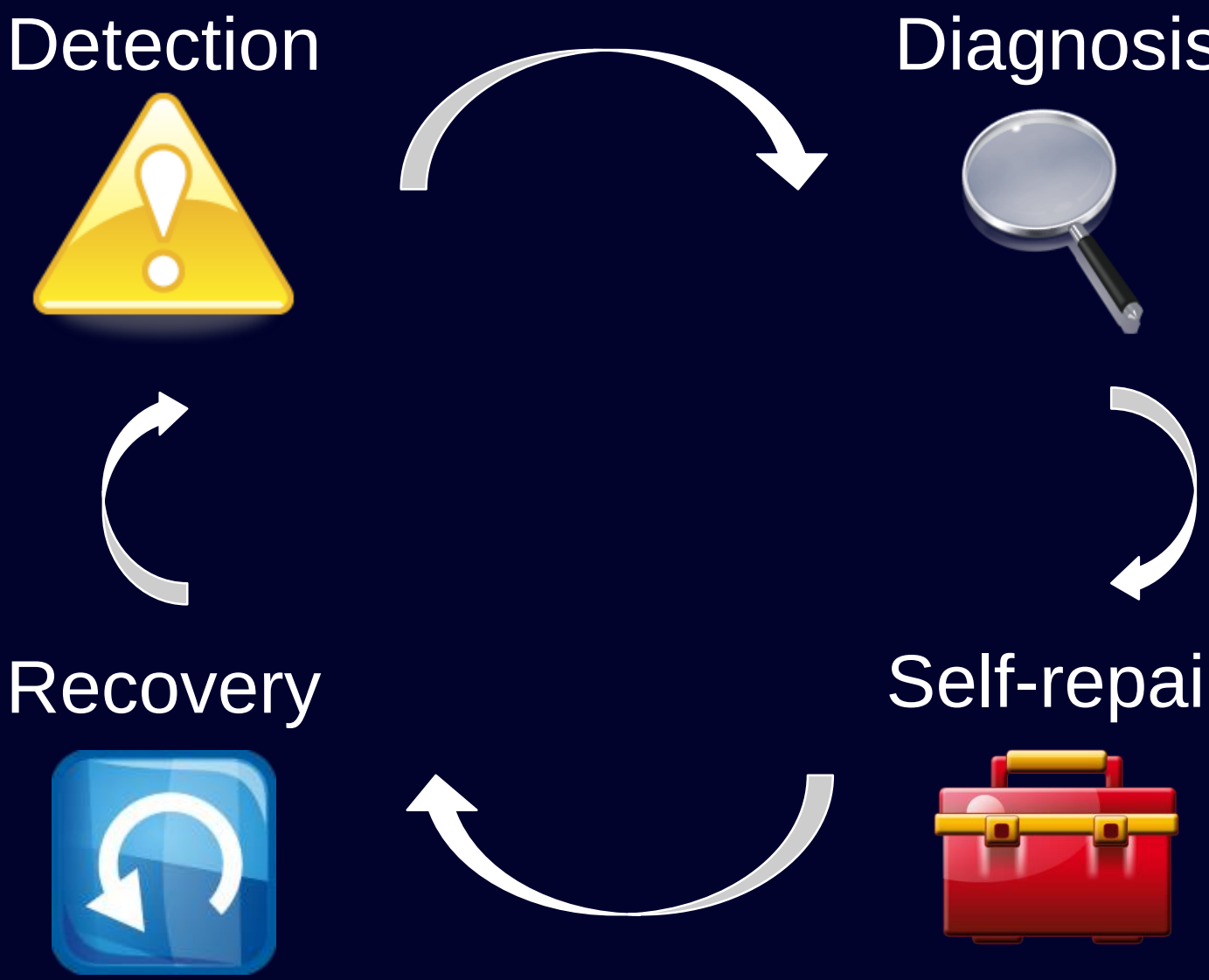
Intel Labs

Stanford University

Hardware Failures: Major Concern

- Permanent: our focus
- Temporary

Tolerating Permanent Hardware Failures



Tolerating Permanent Hardware Failures

Detection

Diagnosis

Concurrent error detection: expensive

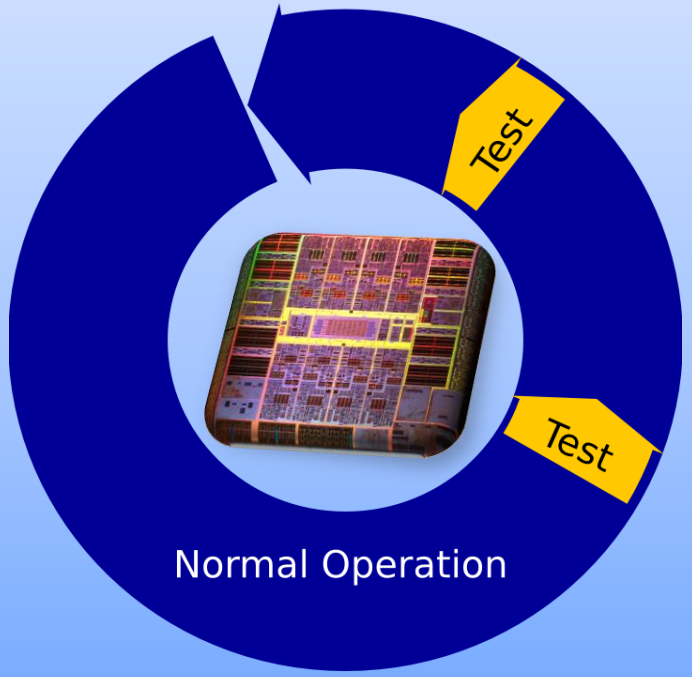
Tolerating Permanent Hardware Failures

Detection

Diagnosis

Online Self-Test and Diagnostics

- Non-continuous
  - ❖ Low power
- Concurrent
  - ❖ No visible downtime



Tolerating Permanent Hardware Failures

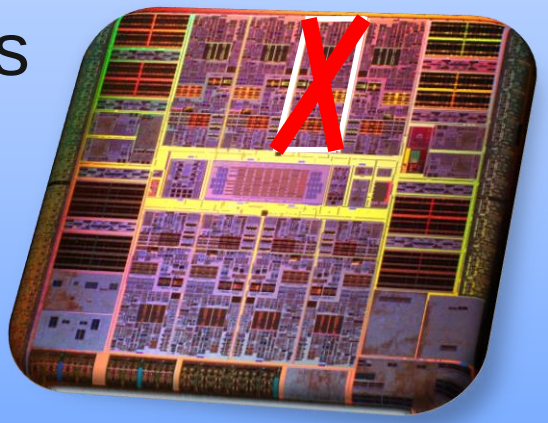
Detection

Diagnosis

Online Self-Test and Diagnostics

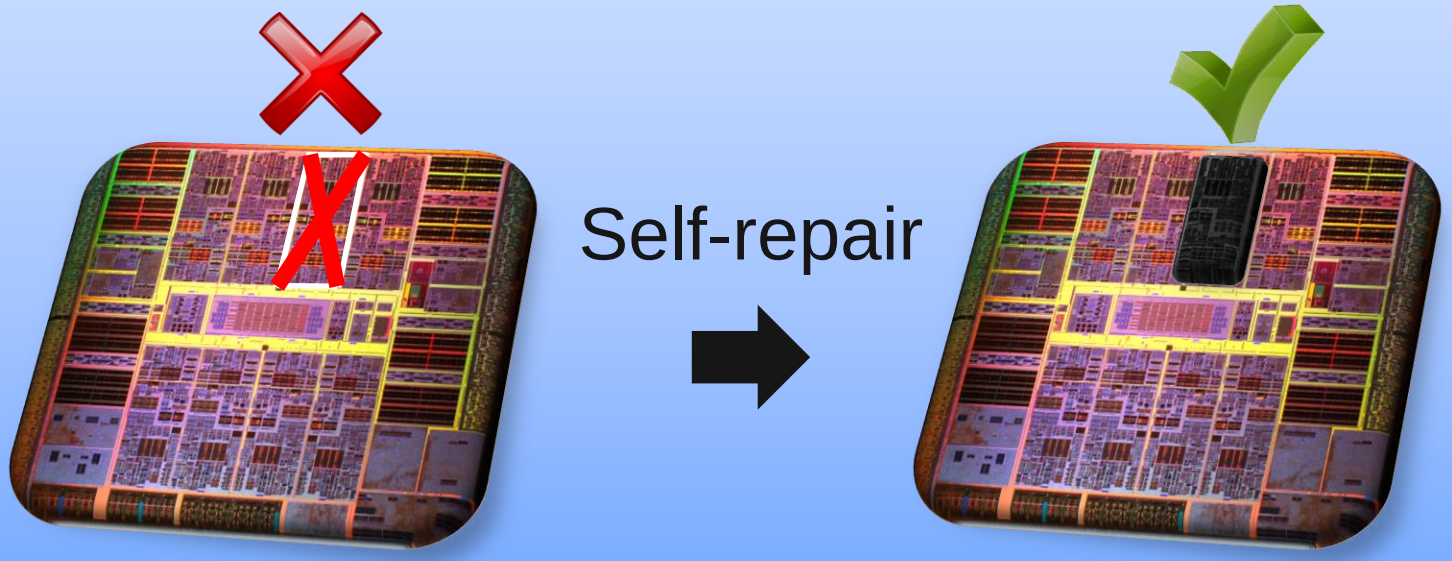
- Localize failures

Diagnosis



Tolerating Permanent Hardware Failures

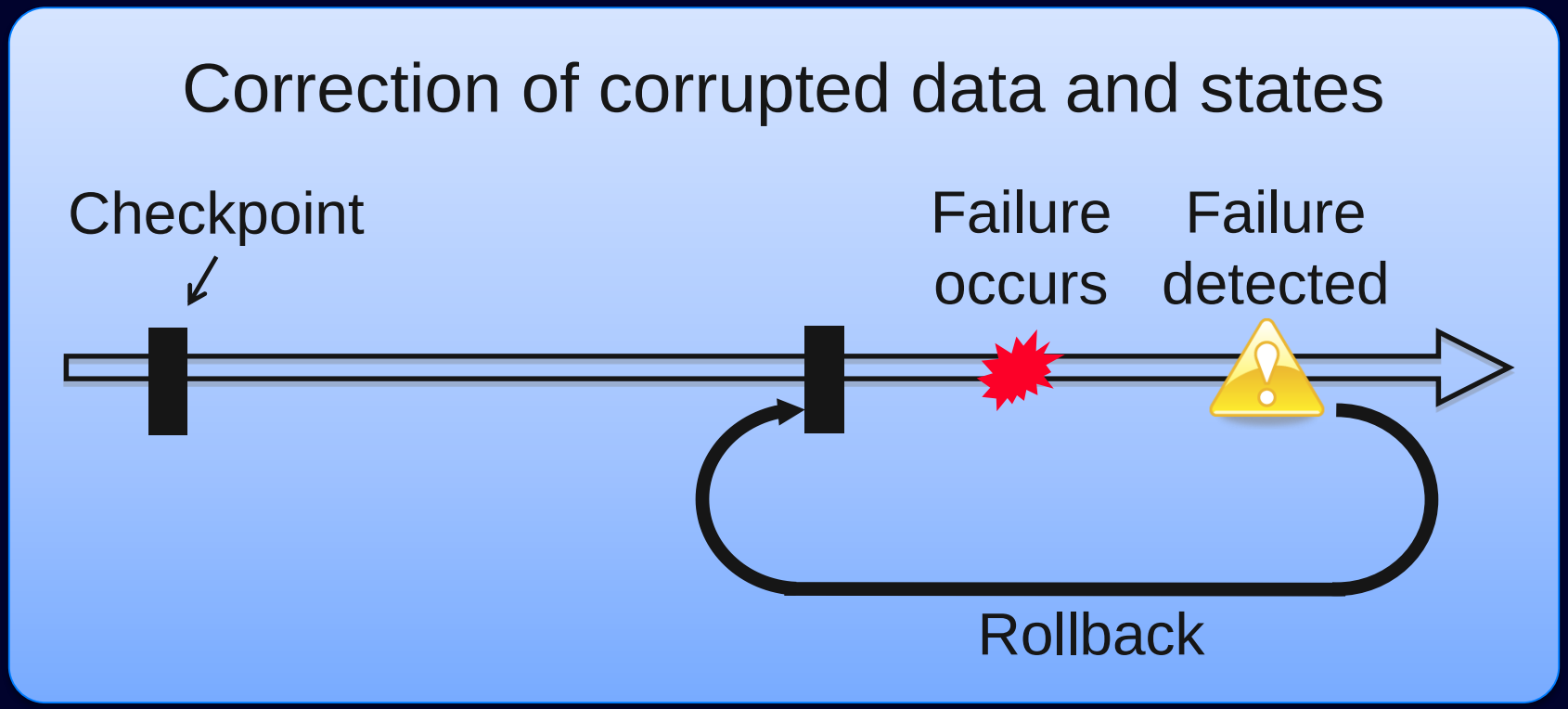
Replace / bypass faulty component(s)



Recovery

Self-repair

Tolerating Permanent Hardware Failures



Recovery

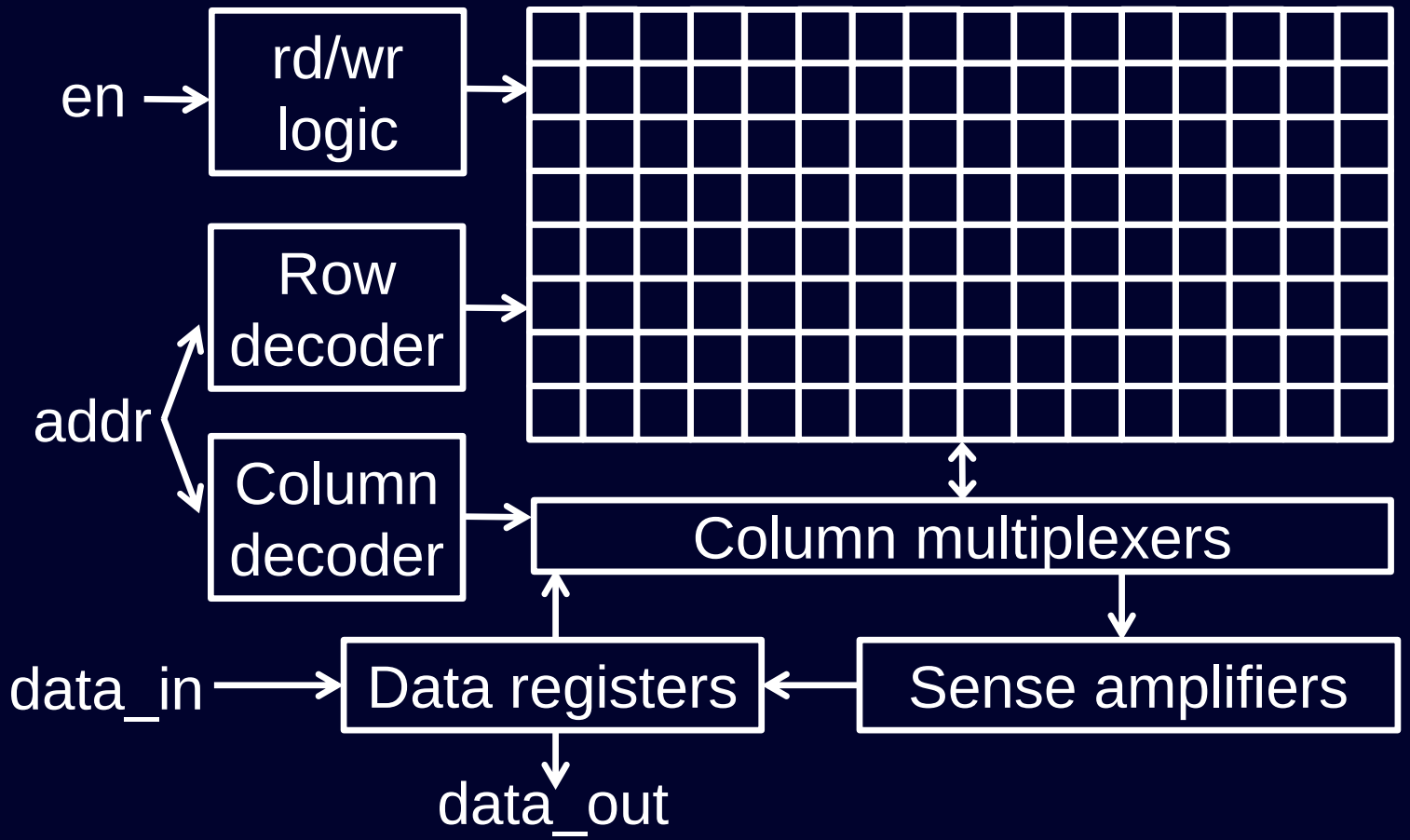
Self-repair

This Lecture

Outline

- Introduction
- Self-repair techniques
  - Memories
  - Processor cores
  - Uncore components
- Conclusion

Memory Organization



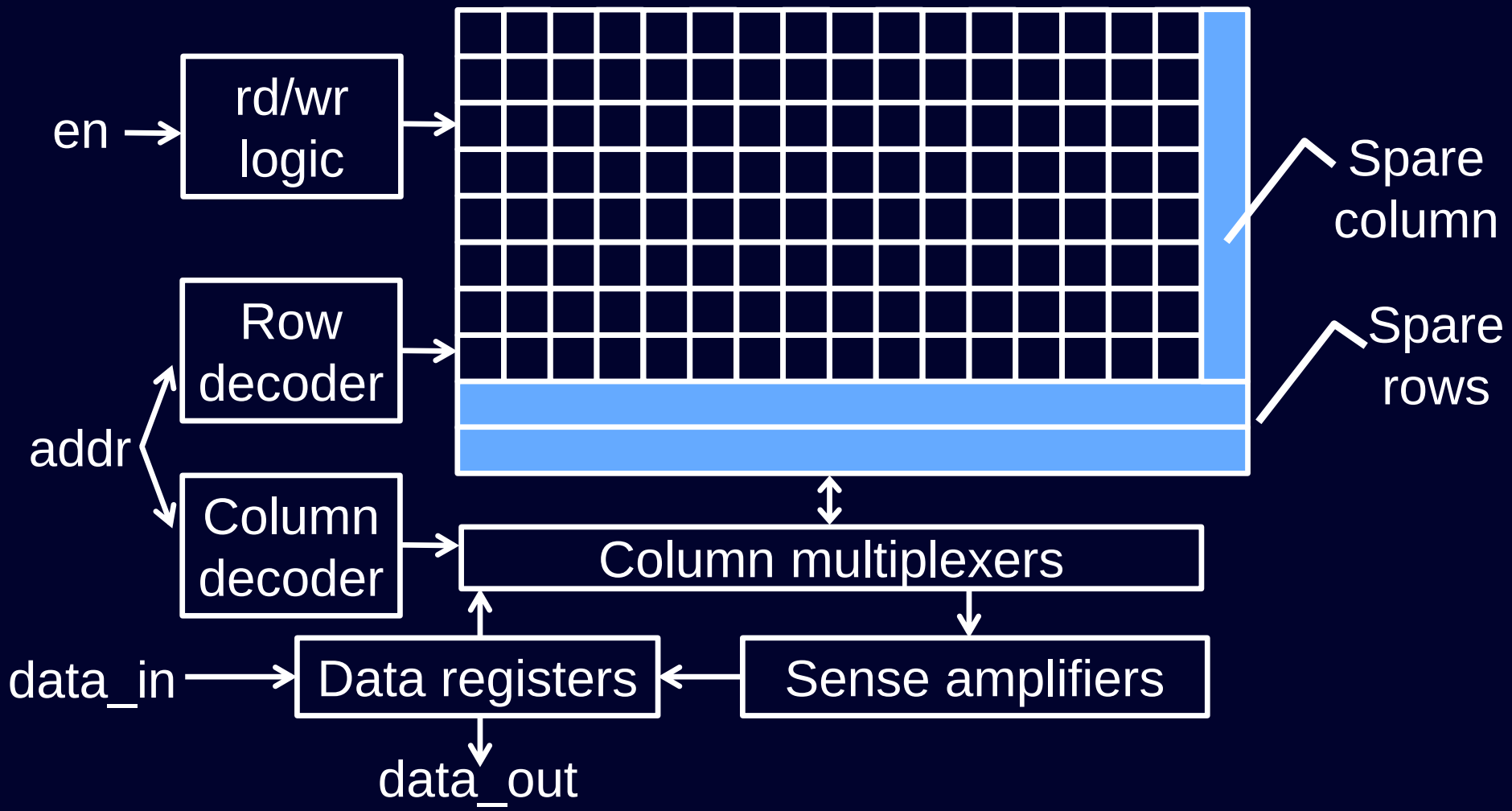
Memory Functional Fault Models

- Memory cell array faults
  - stuck-at, transition, coupling, etc.
- Address decode faults (AFs)
  - no cell accessed, multiple cells accessed, etc.
- Read/write logic faults
  - Equivalent to memory cell array faults

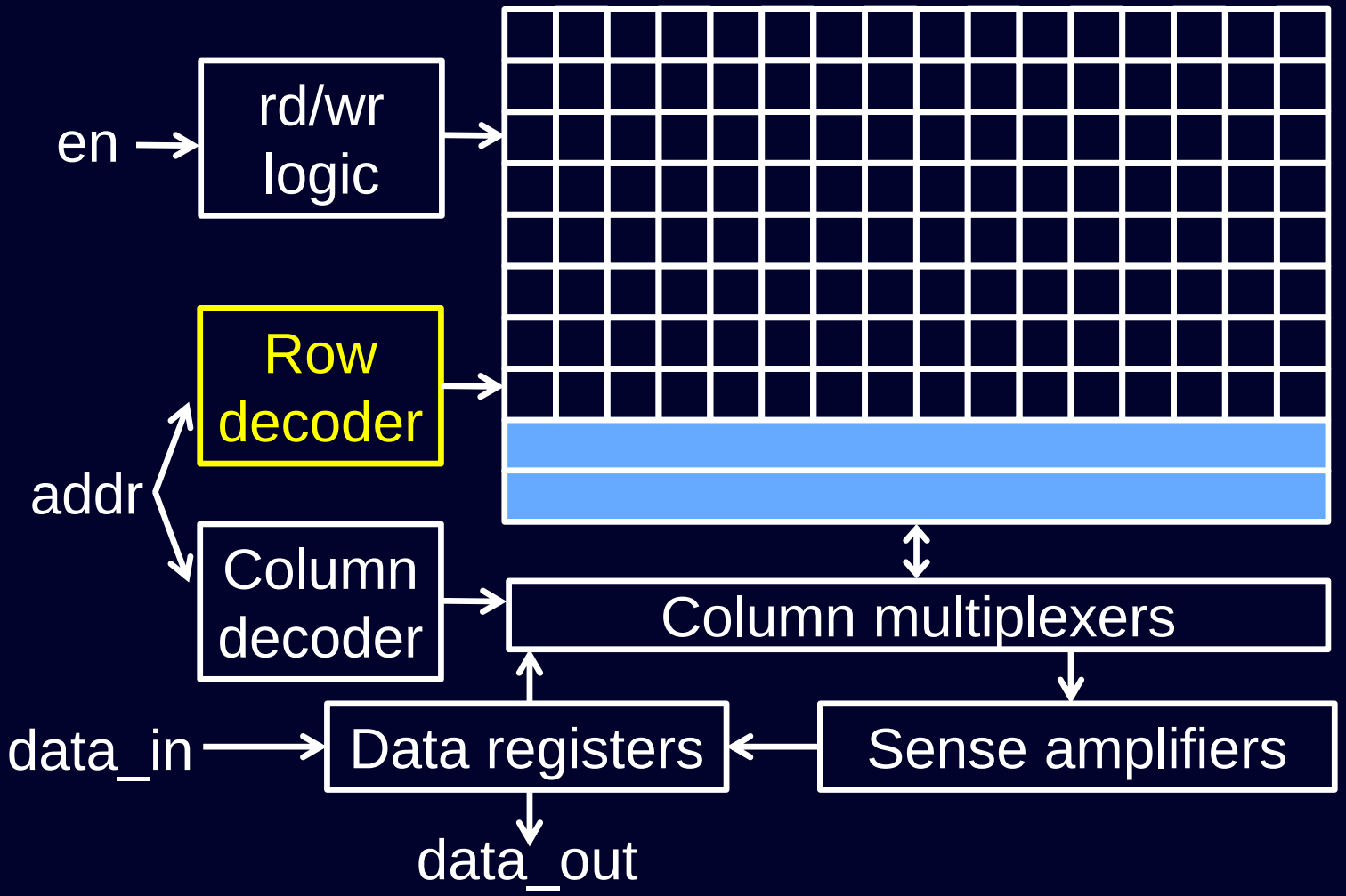
Single bit / word / column / row faults dominate

[Aitken 04, Kurdahi 06, Sridharan 12]

Spare Rows / Columns

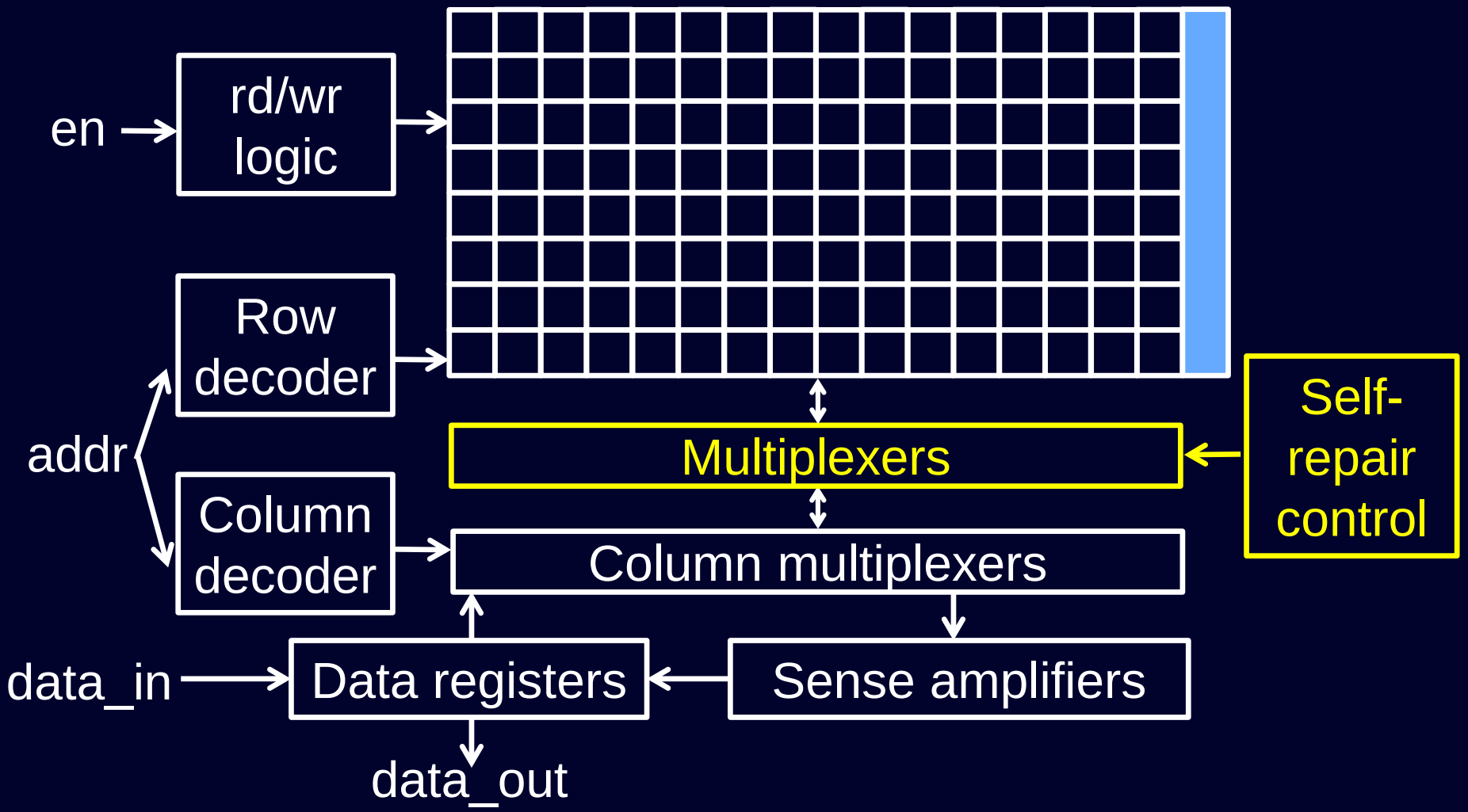


Memory Self-Repair Using Spare Rows



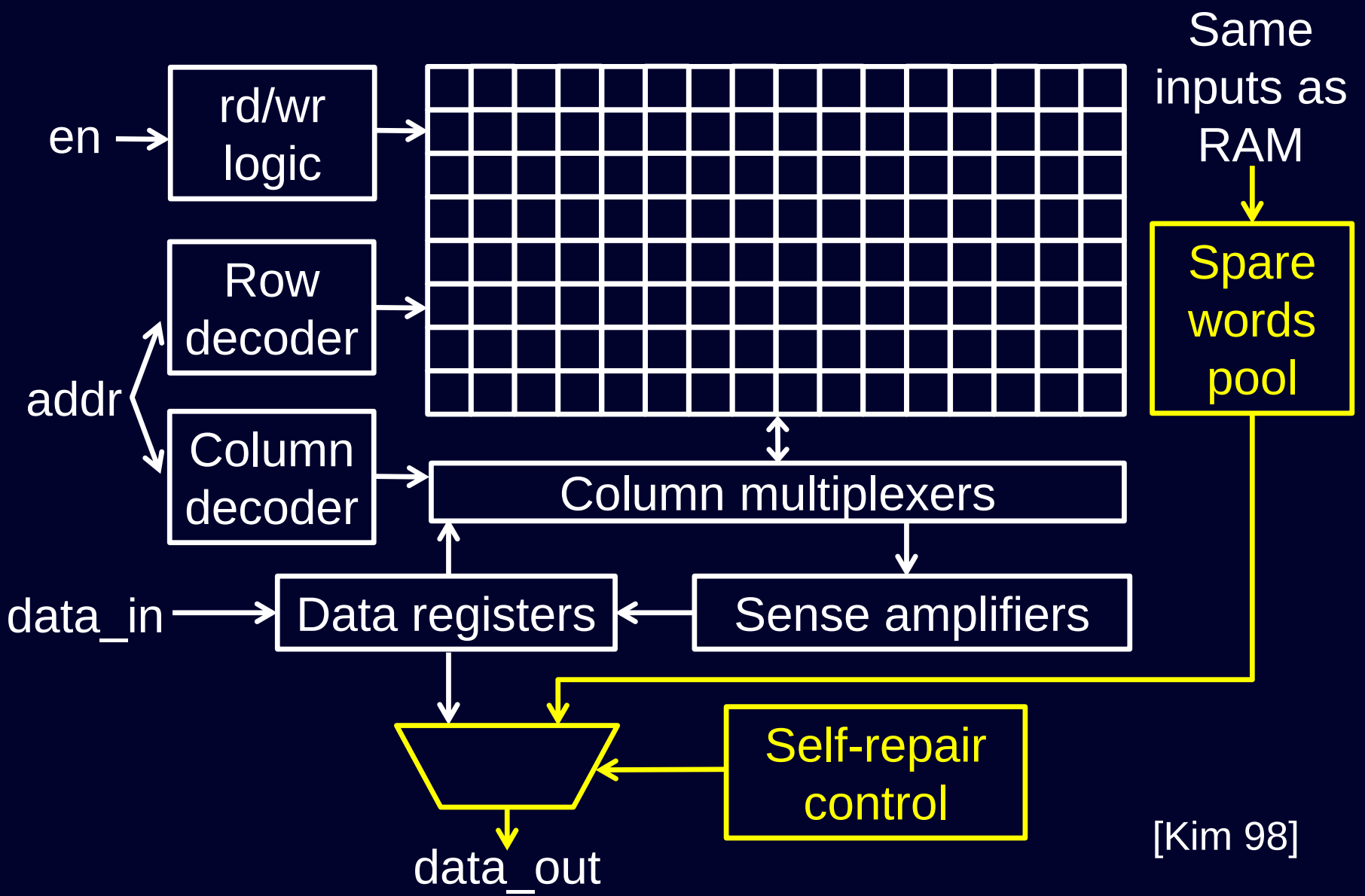
- Row decoder modified

Memory Self-Repair Using Spare Columns



- Additional multiplexers and select logic

Spare Words



[Kim 98]

Redundancy in Memories

- Essential to improve yield and reliability
- Widely used in commercial RAMs
- Related topics
  - How much redundancy?
  - Built-in repair analysis: redundancy allocation

How Much Redundancy?

- Yield analysis and yield learning
- Example: negative binomial memory yield model

$$\square\square\square\square\square = (1 + \square\square_{\ddot{u}}\square)$$

$\square$: defect desntiy, $\square$: memory area

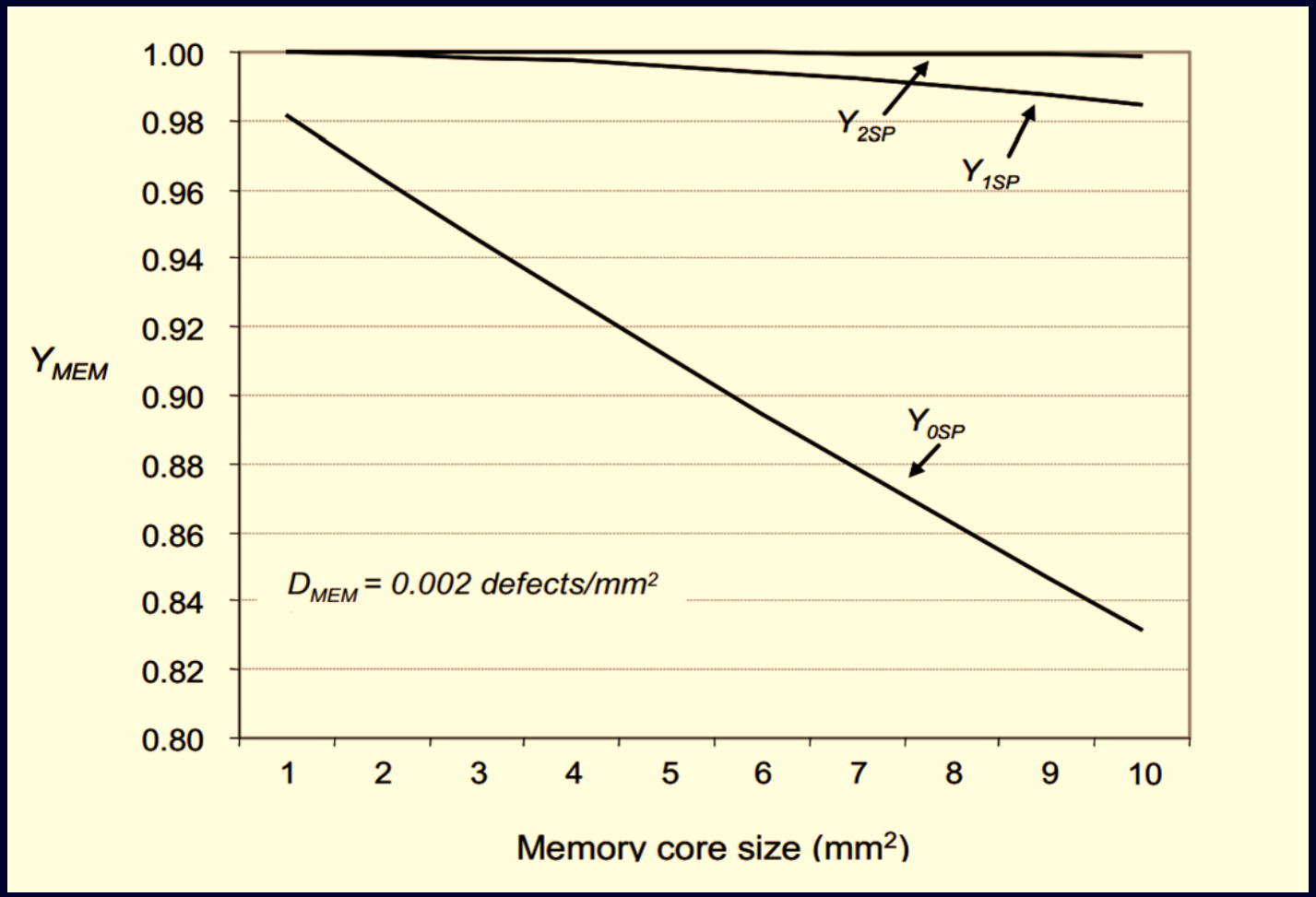
$\square$: defect clustering coefficient (measured to be 2 or 3)

- For a memory array with N rows and 1 spare row

$$\square\square\square\square\square = \square\square\square\square\square+(\square + 1)(\square\square\square\square\square)(1 - \square\square\square\square\square)$$

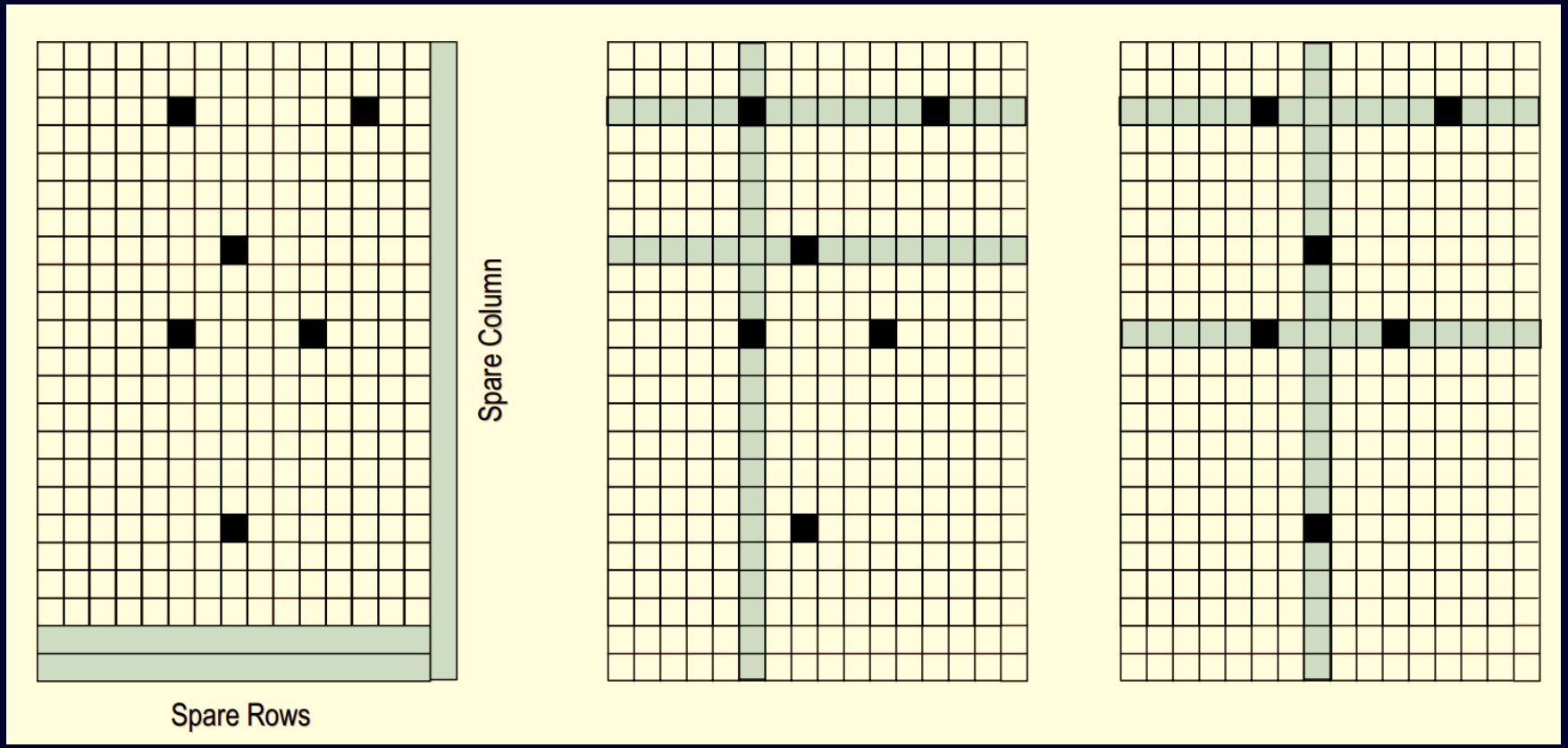
How Much Redundancy?

- Yield analysis and yield learning
- Example: negative binomial memory yield model



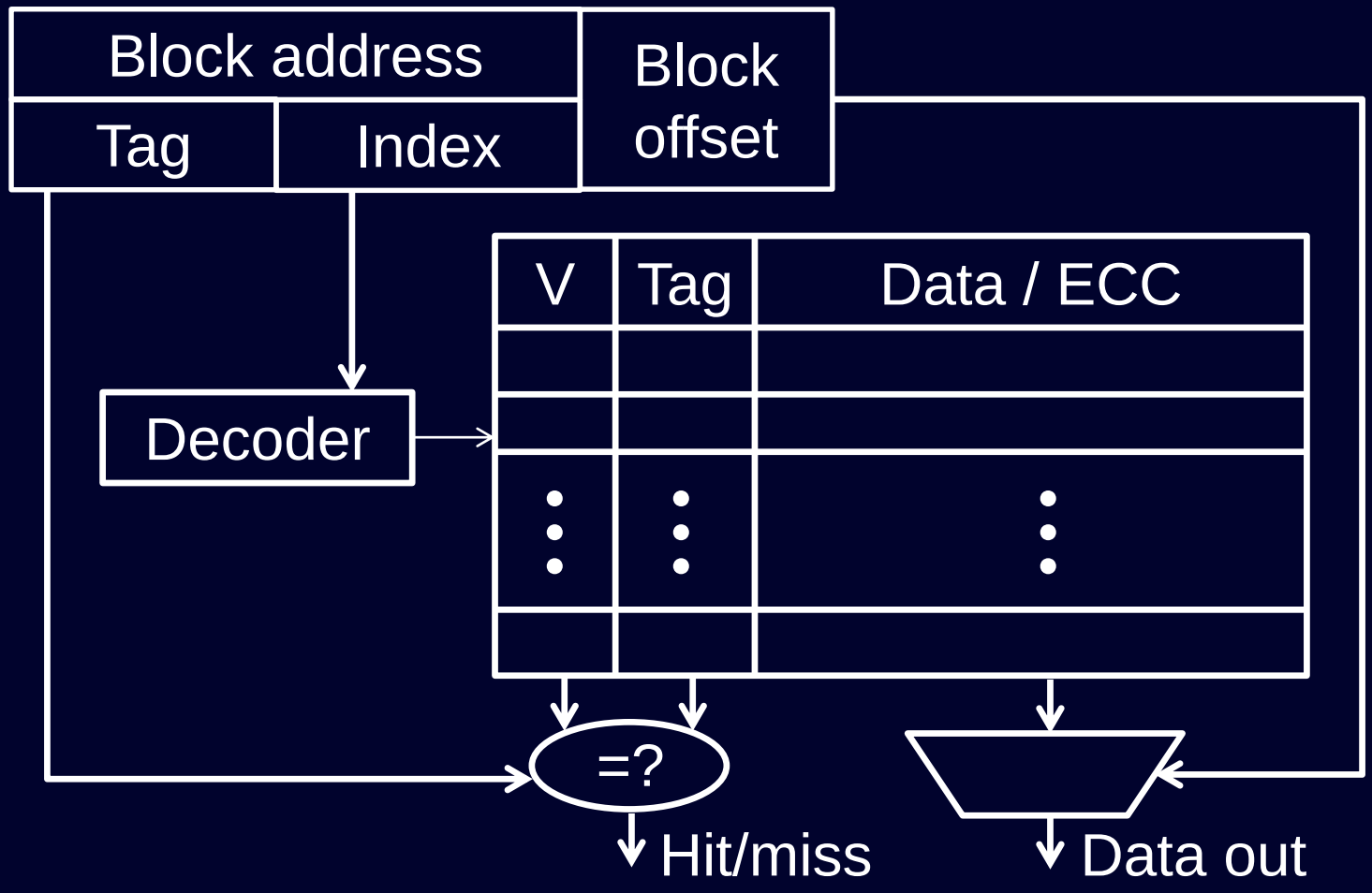
Built-In Repair Analysis

- Spare rows and columns
  - NP complete
  - Various algorithms and EDA tools exist



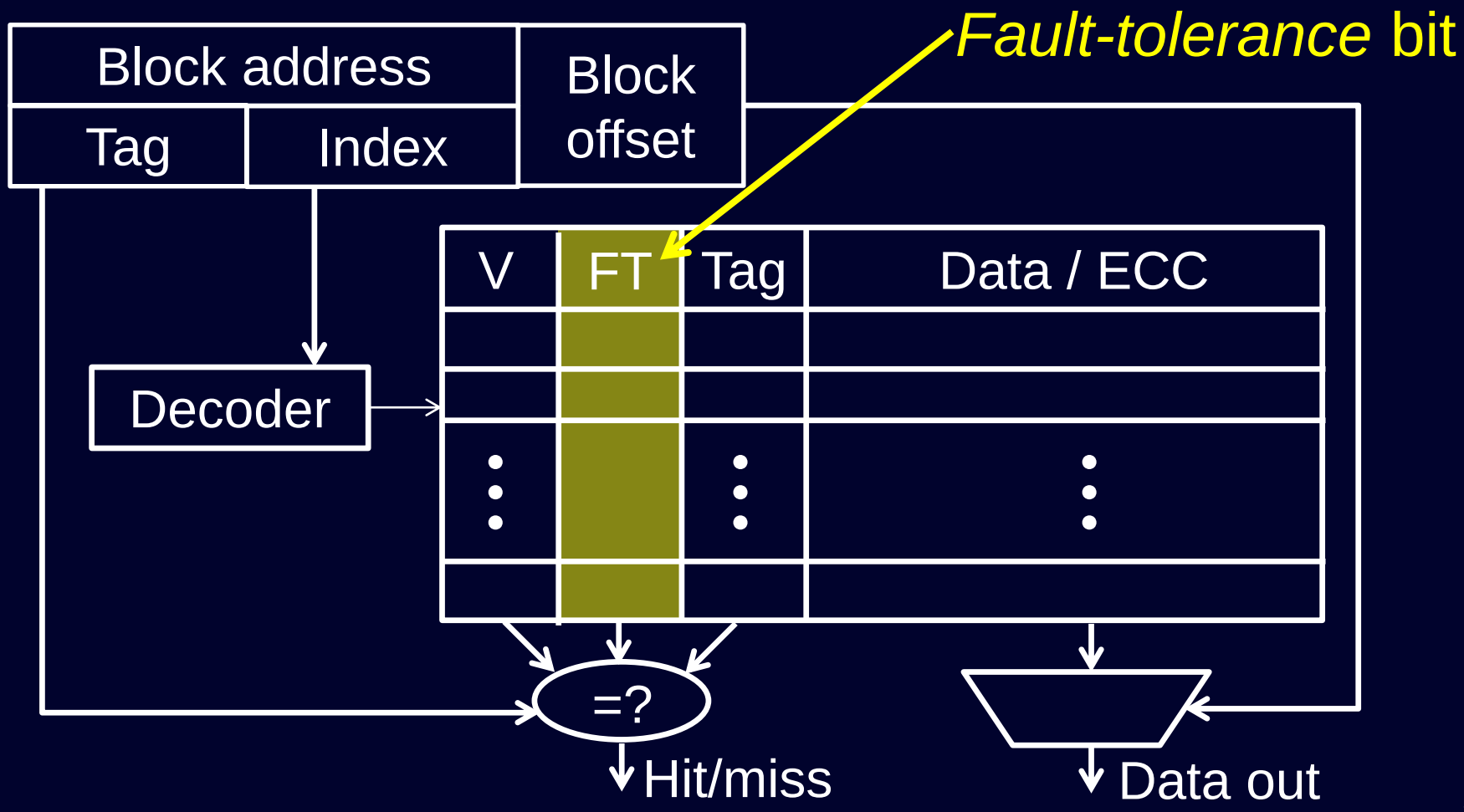
Special Self-Repair Techniques for Caches

- Caches: affects performance, NOT functionality

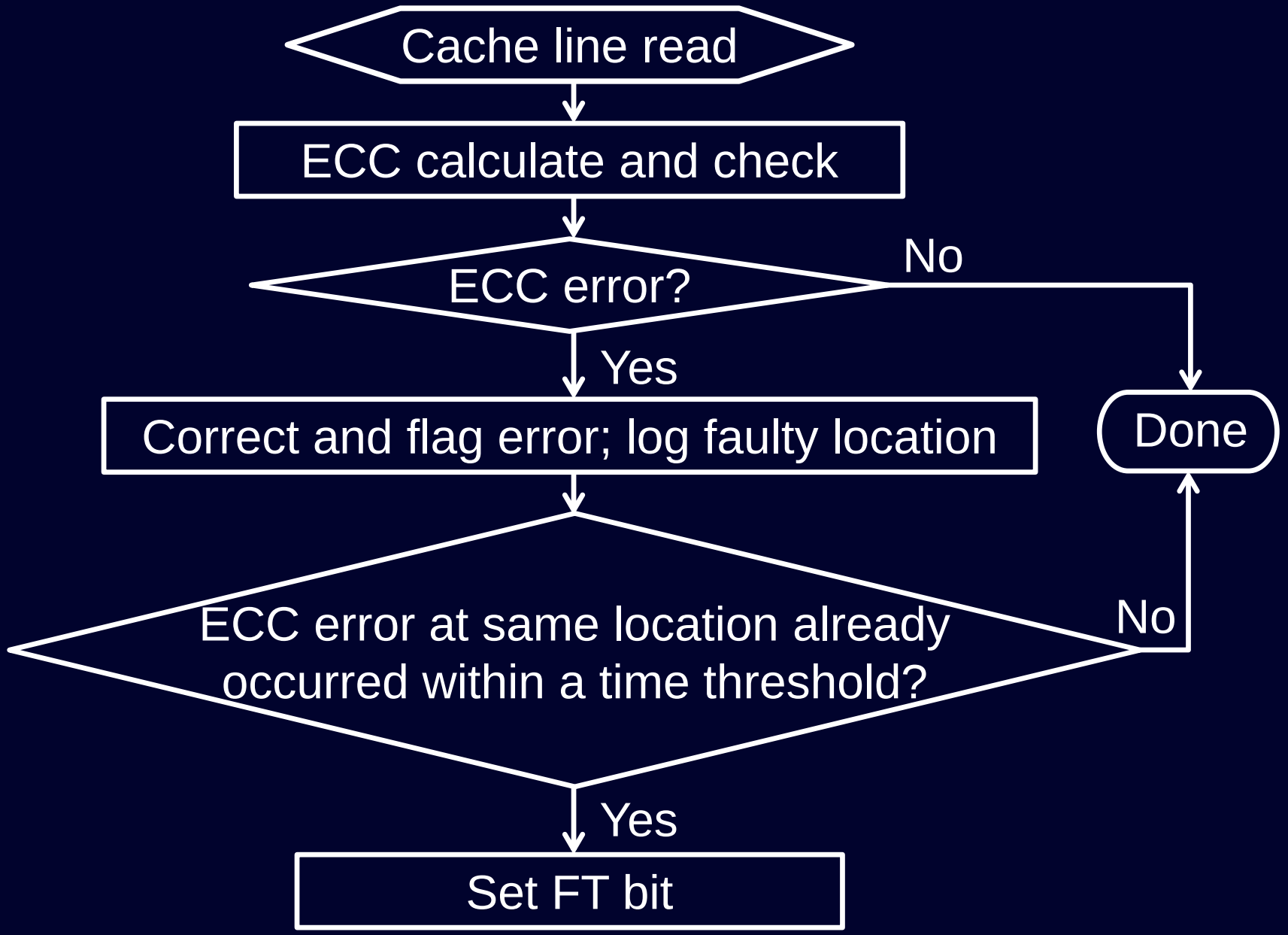


Cache Line Disable/Delete

- Exist in commercial systems
  - Intel [Chang 07], IBM [Sanda 08], etc.



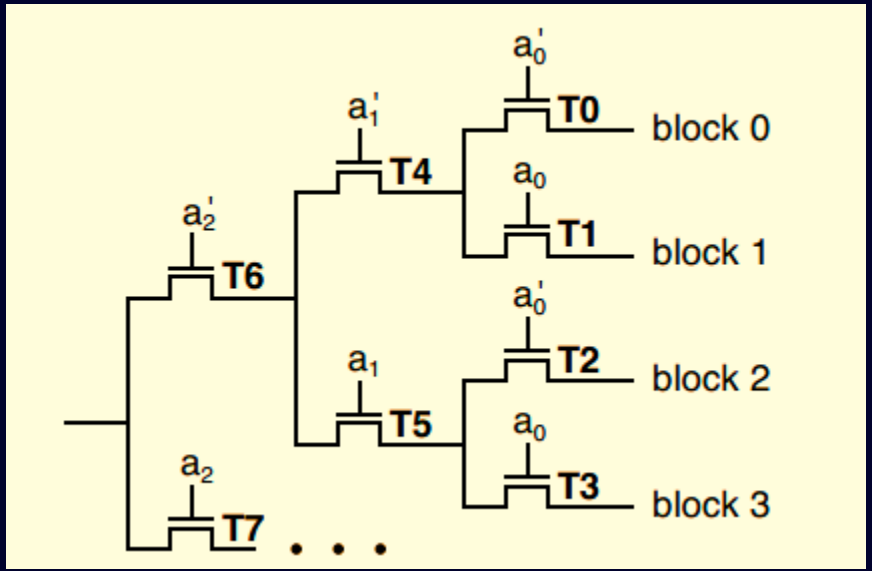
Setting the FT Bit



PADded Cache

- Reconfigurable cache with programmable decoder

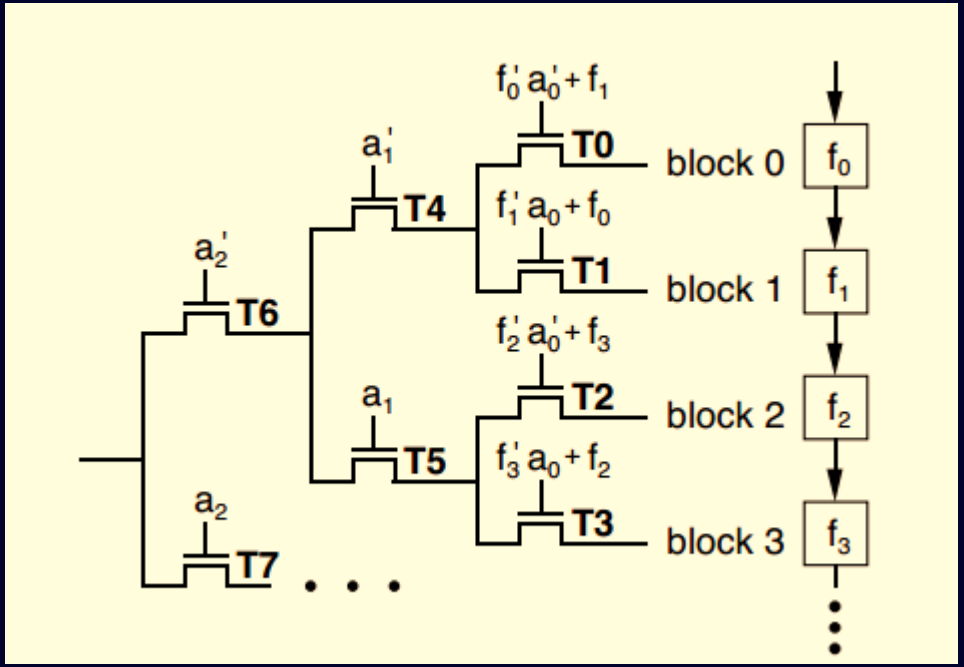
Conventional decoder



PADded Cache

- Reconfigurable cache with programmable decoder
  - Additional tag bits needed

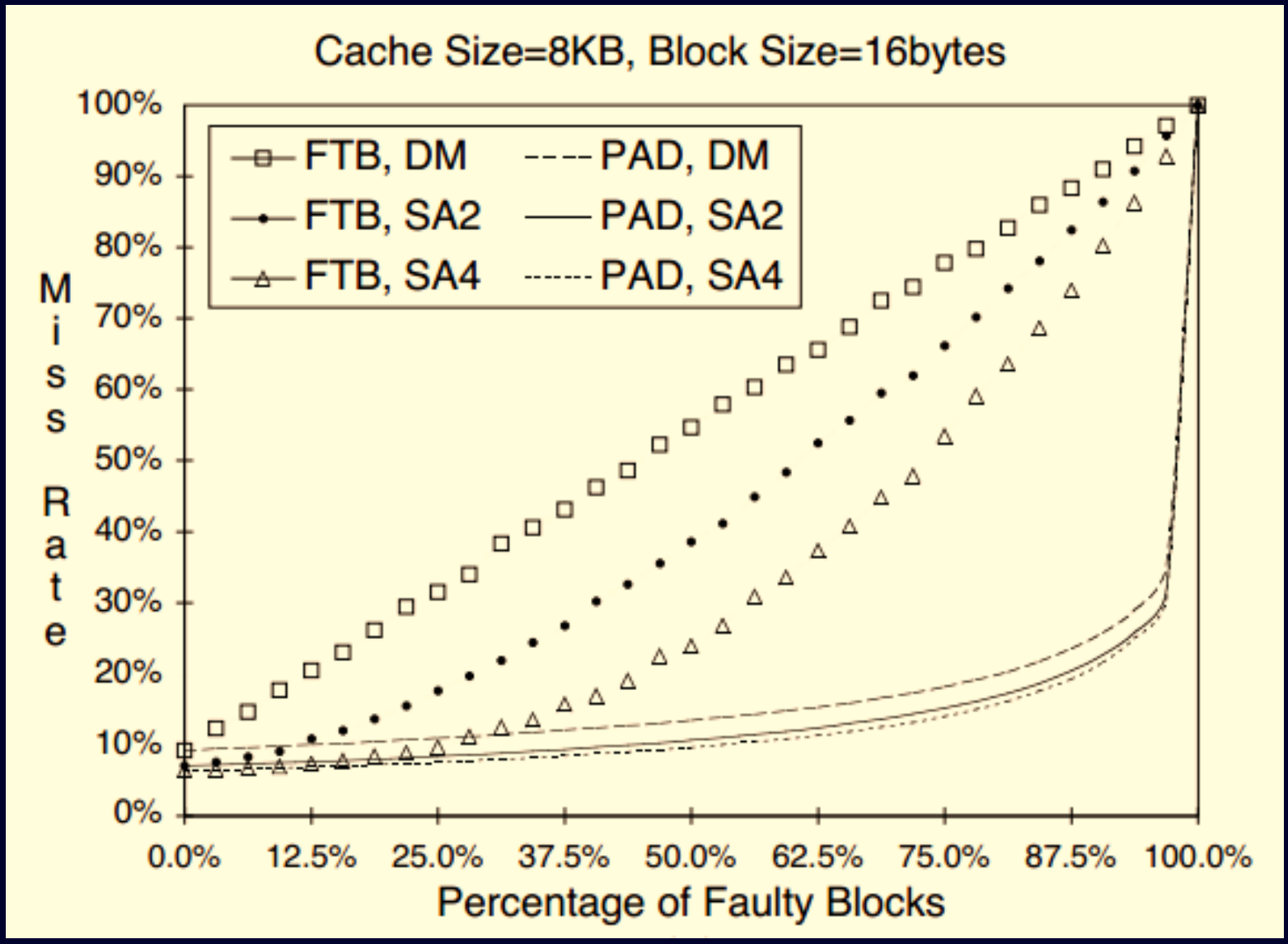
Programmable decoder



[Shirvani 99]

- ~ 5% area cost
  - 16KB, direct-mapped, 1-level programmability
  - Reduce cost at the price of granularity

Cache Line Delete vs. PADded Cache



Outline

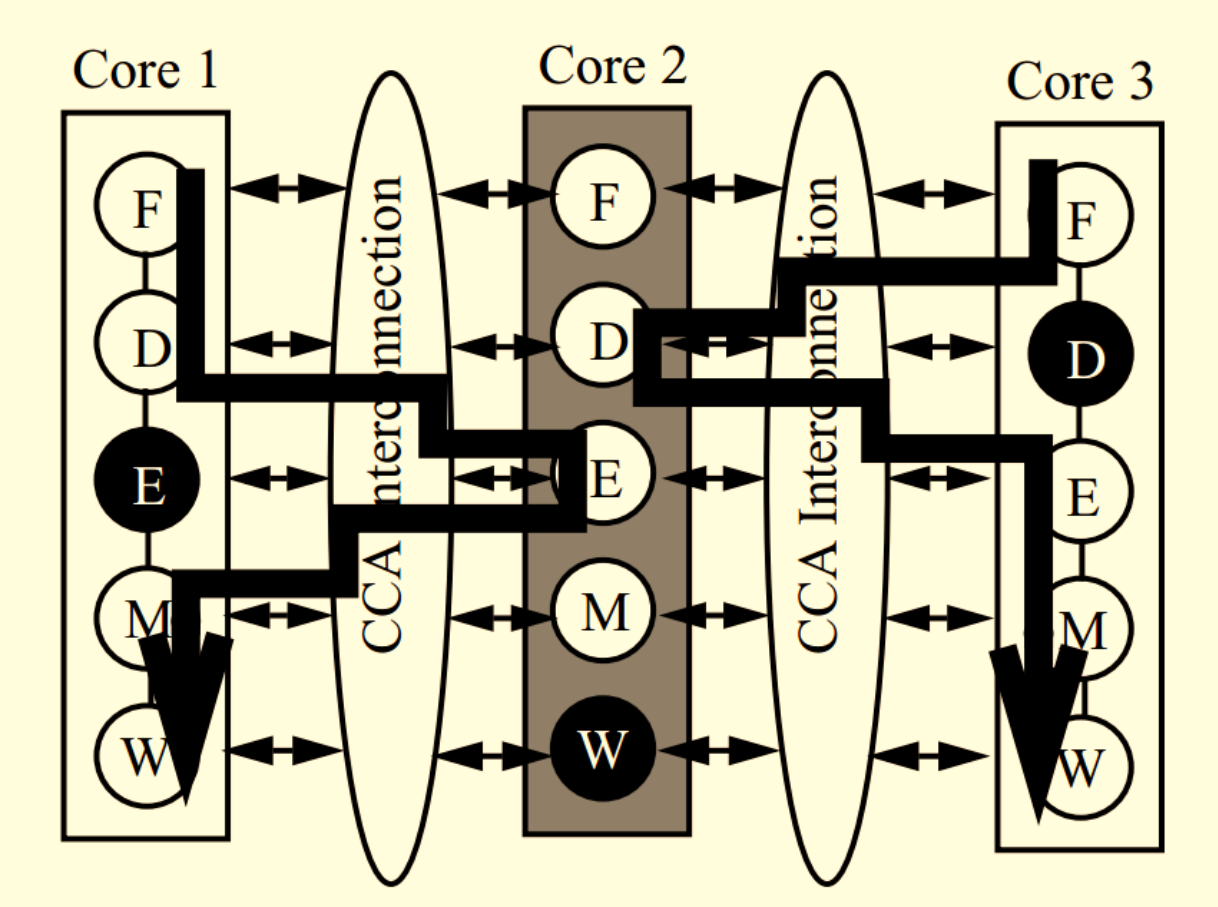
- Introduction
- Self-repair techniques
  - Memories
  - Processor cores
  - Uncore components
- Conclusion

Core Sparing

- Utilize multi-/many-core architectures
  - Core disabling also possible
- Already in commercial products
  - IBM BlueGene/Q
  - Nvidia Geforce
  - Cisco Metro
- Fine-grained approaches?

Core Cannibalization

- Many-core designs with small in-order cores
  - 3.5% area cost (OpenRISC 1200)



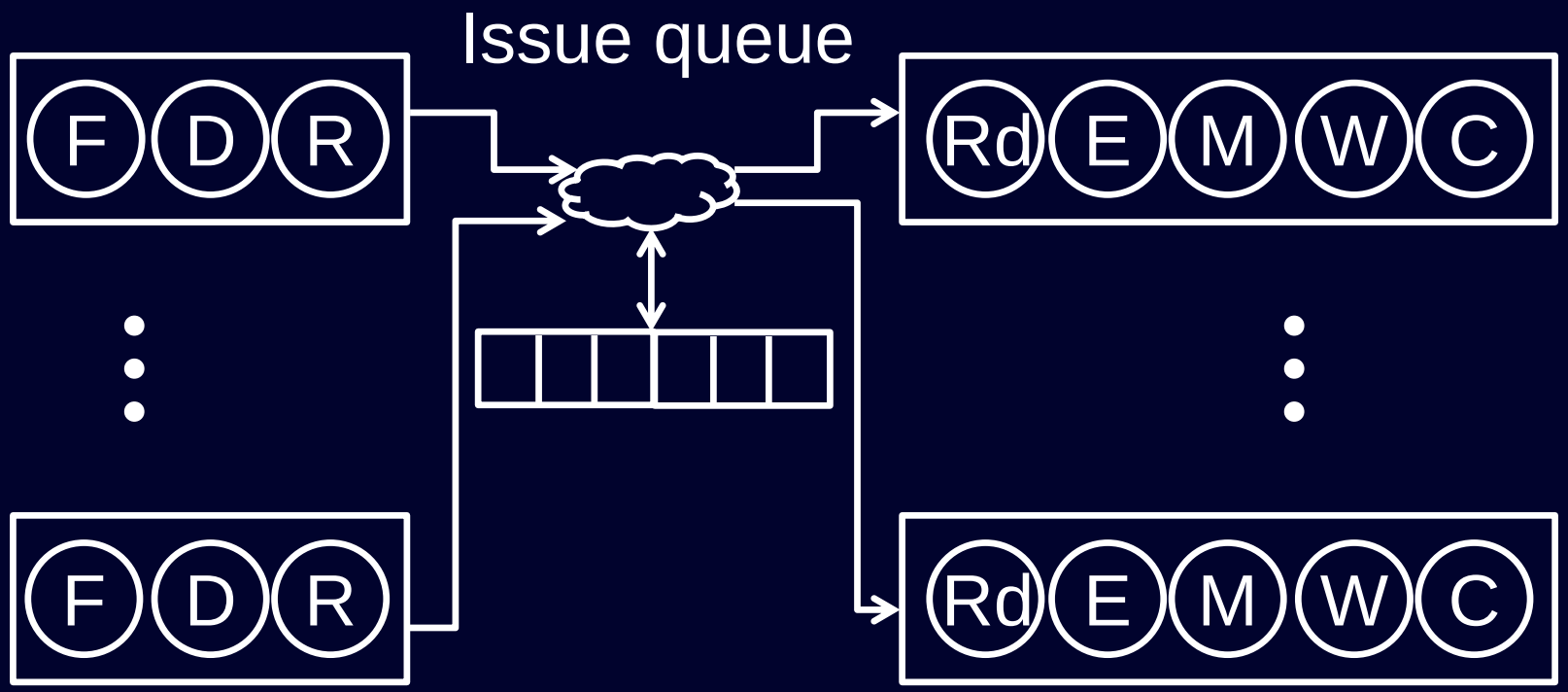
[Romanescu 08]

Core Cannibalization: Discussion

- What about diagnosis?
- Routing → performance impact
  - Additional wire delay pipeline stages
    - Modified branch resolution, bypass logic, etc.
  - Decreased clock frequency
- Small vs. large number of faulty cores

Microarchitectural Block Disabling

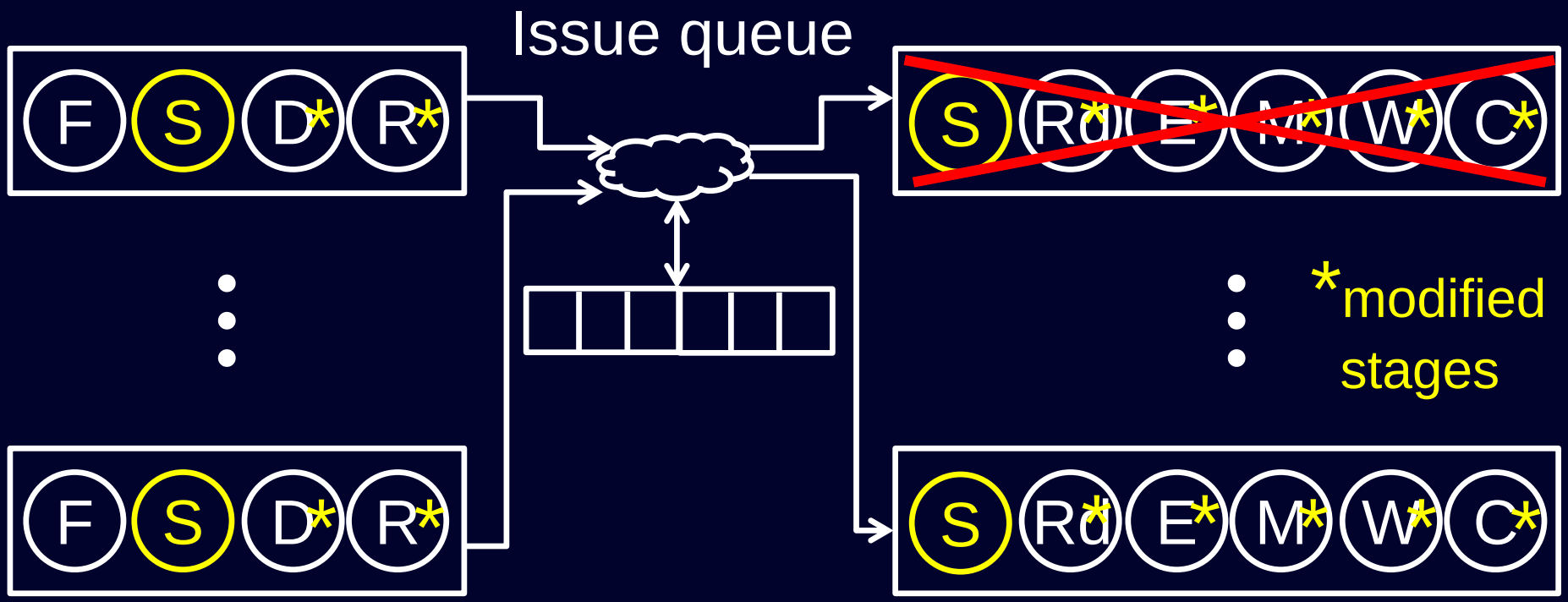
- Disable “half pipeline way” in superscalar designs
  - 12% area cost (includes diagnosis)



[Schuchman 05]

Microarchitectural Block Disabling

- Disable “half pipeline way” in superscalar designs
  - 12% area cost (includes diagnosis)

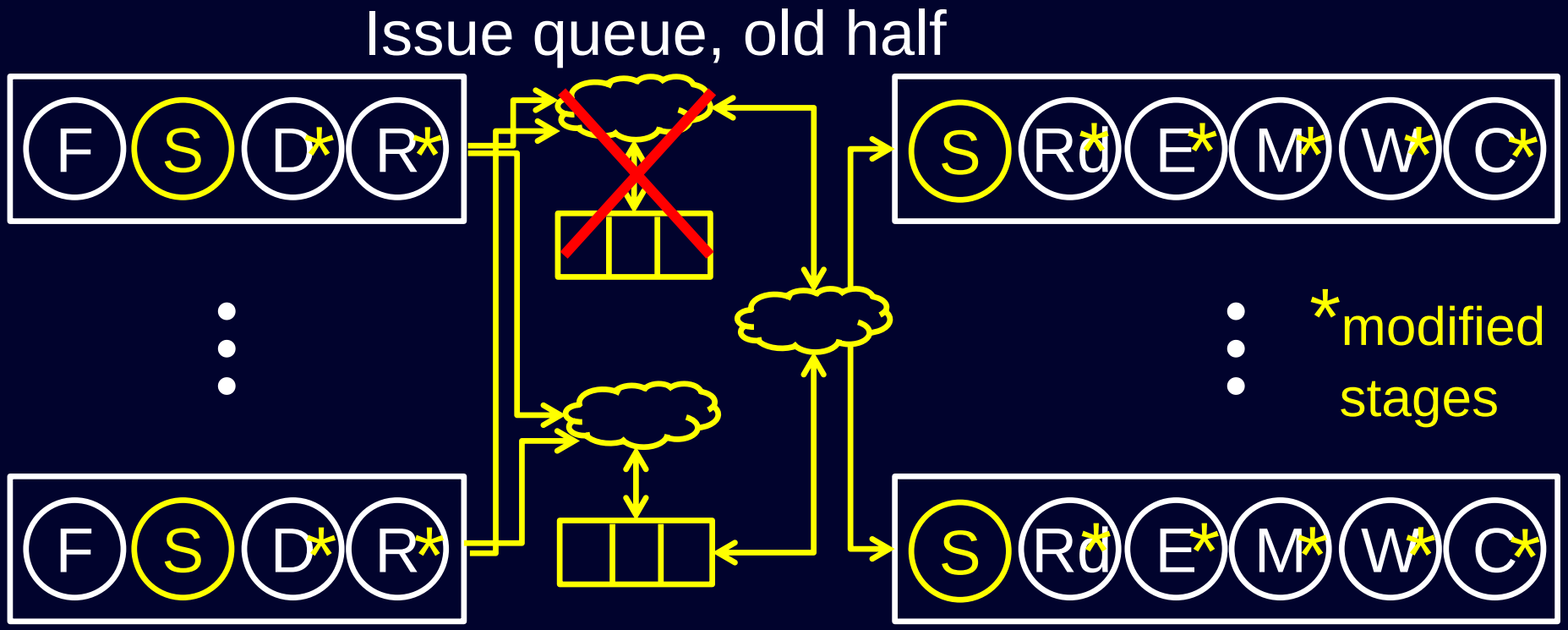


[Schuchman 05]

2 shift stages (S) added and lots design modifications

Microarchitectural Block Disabling

- Disable “half pipeline way” in superscalar designs
  - 12% area cost (includes diagnosis)



[Schuchman 05]

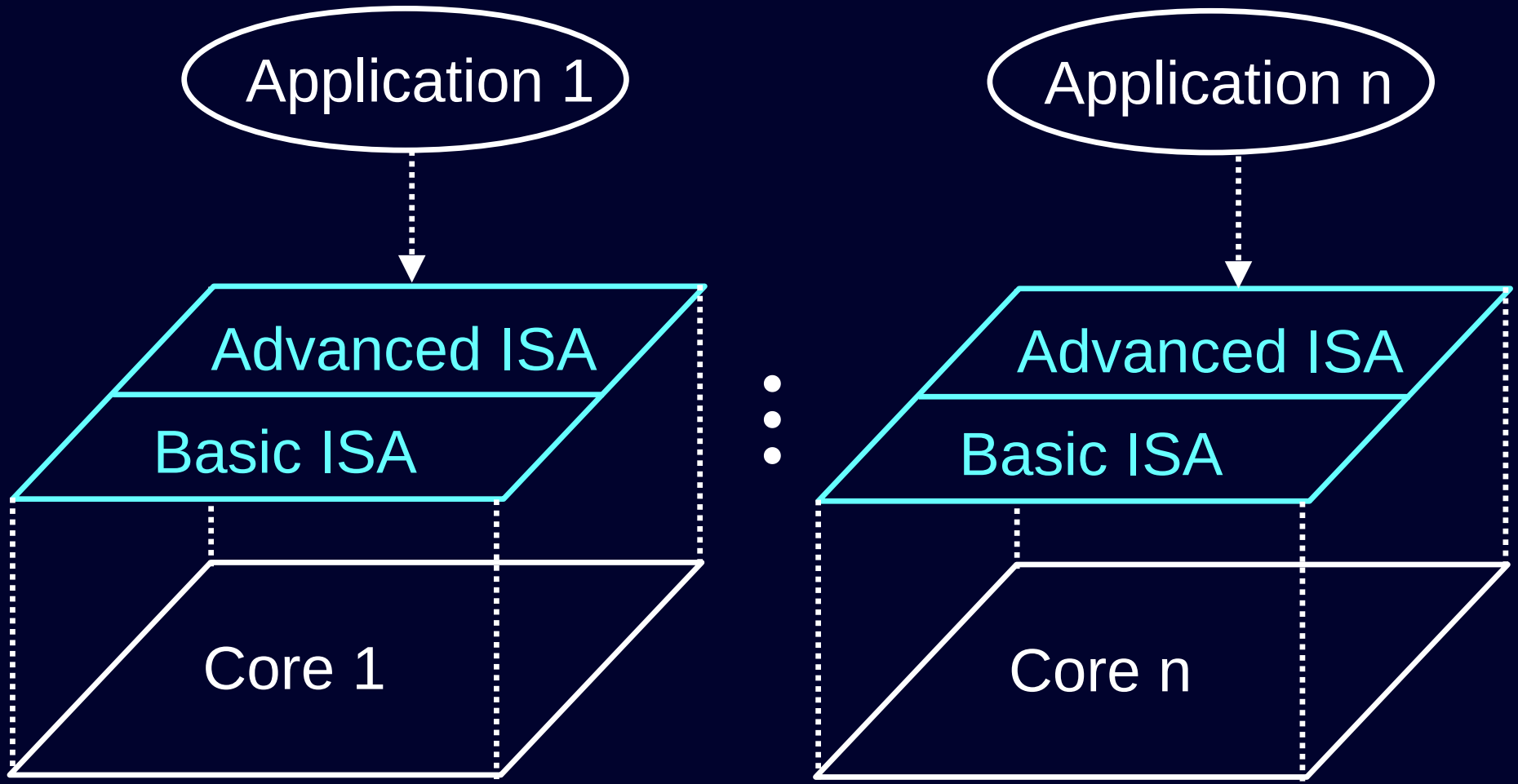
Split issue queue (same for store buffer, not shown)

Microarchitectural Block Disabling: Discussion

- Expensive, complex, intrusive
  - Diagnosis logic
  - Reconfiguration logic
- Coverage issues
  - E.g., fetch stage not covered

Architectural Core Salvaging

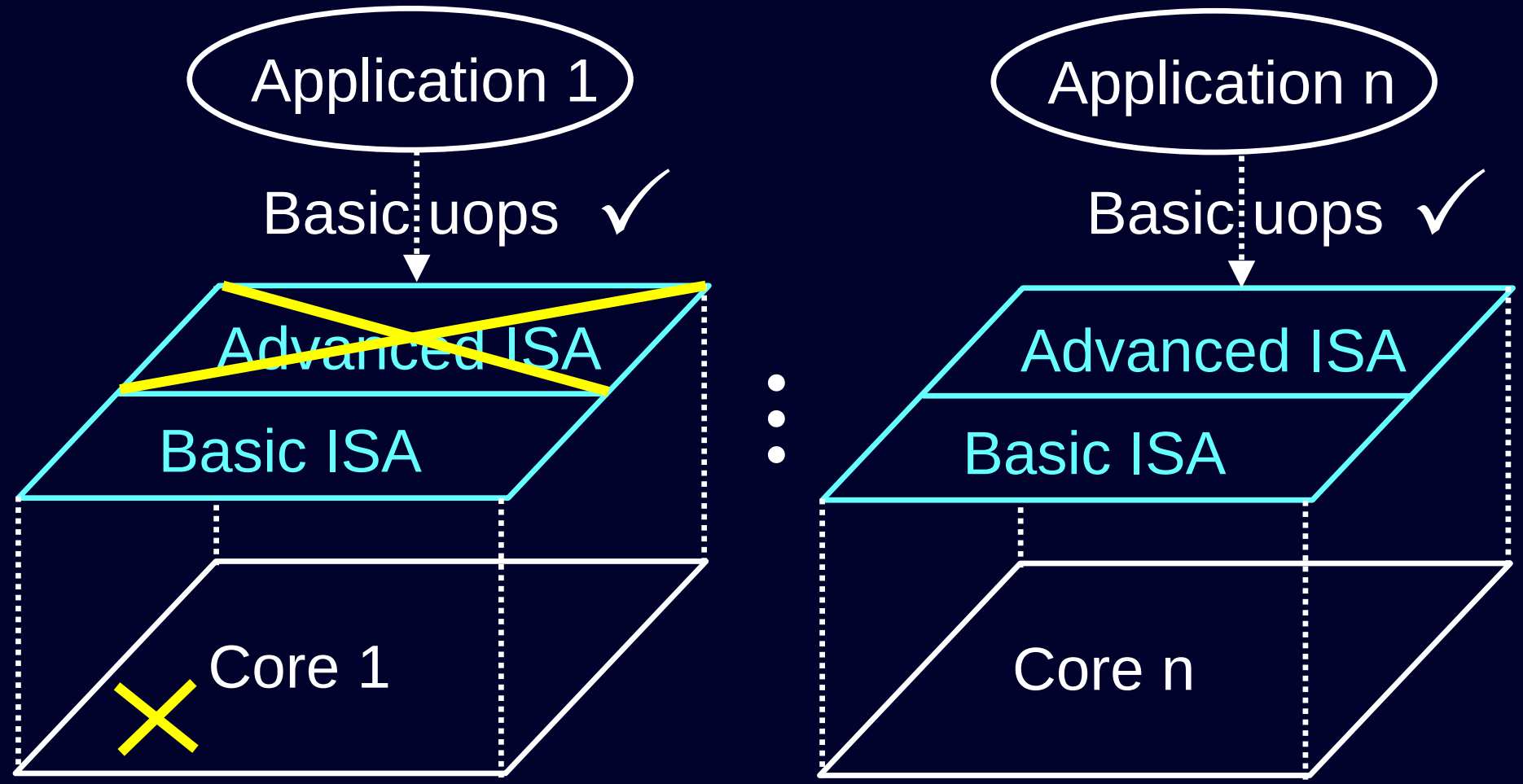
- Many-core CISC processor designs



[Powell 09]

Architectural Core Salvaging

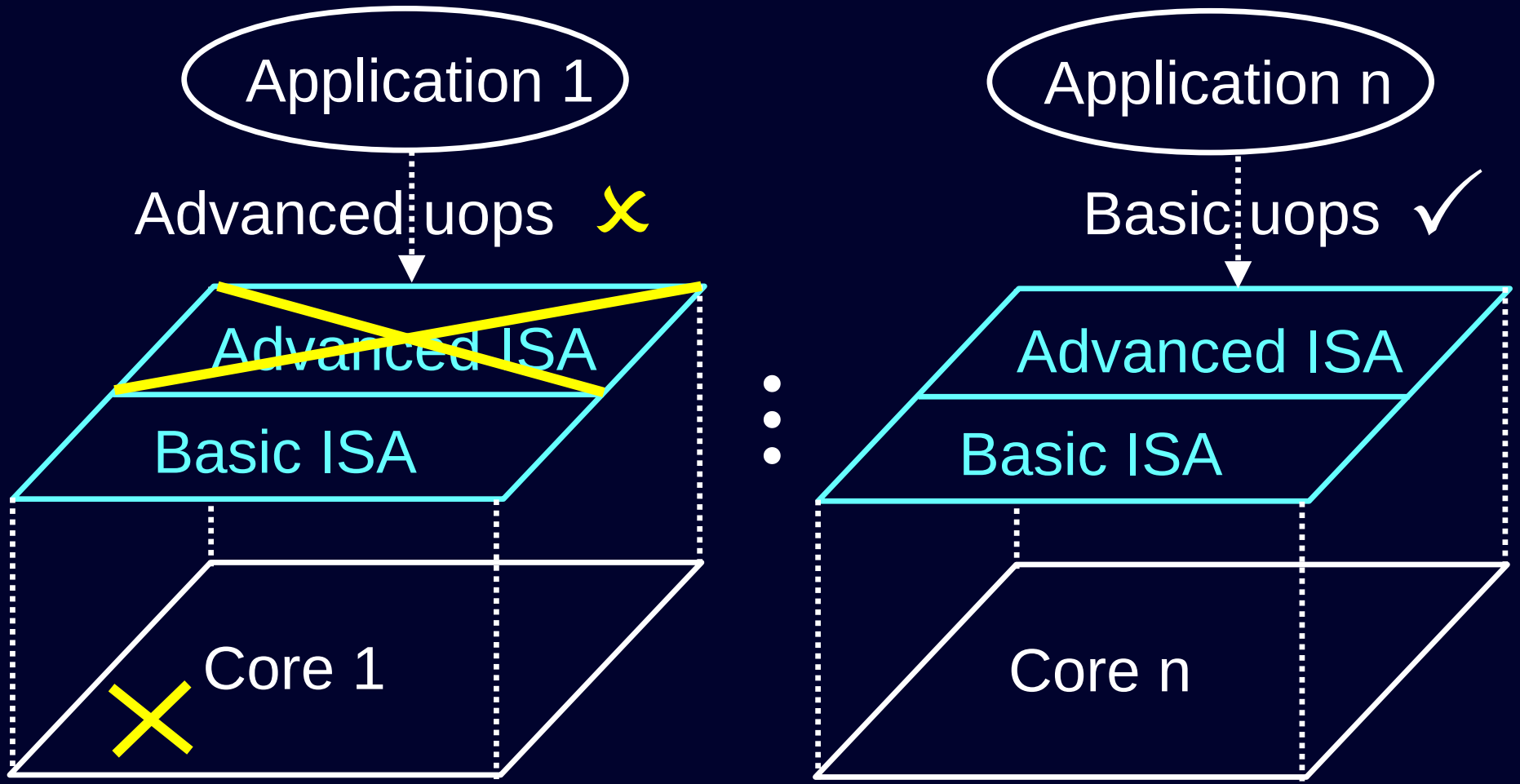
- Many-core CISC processor designs



[Powell 09]

Architectural Core Salvaging

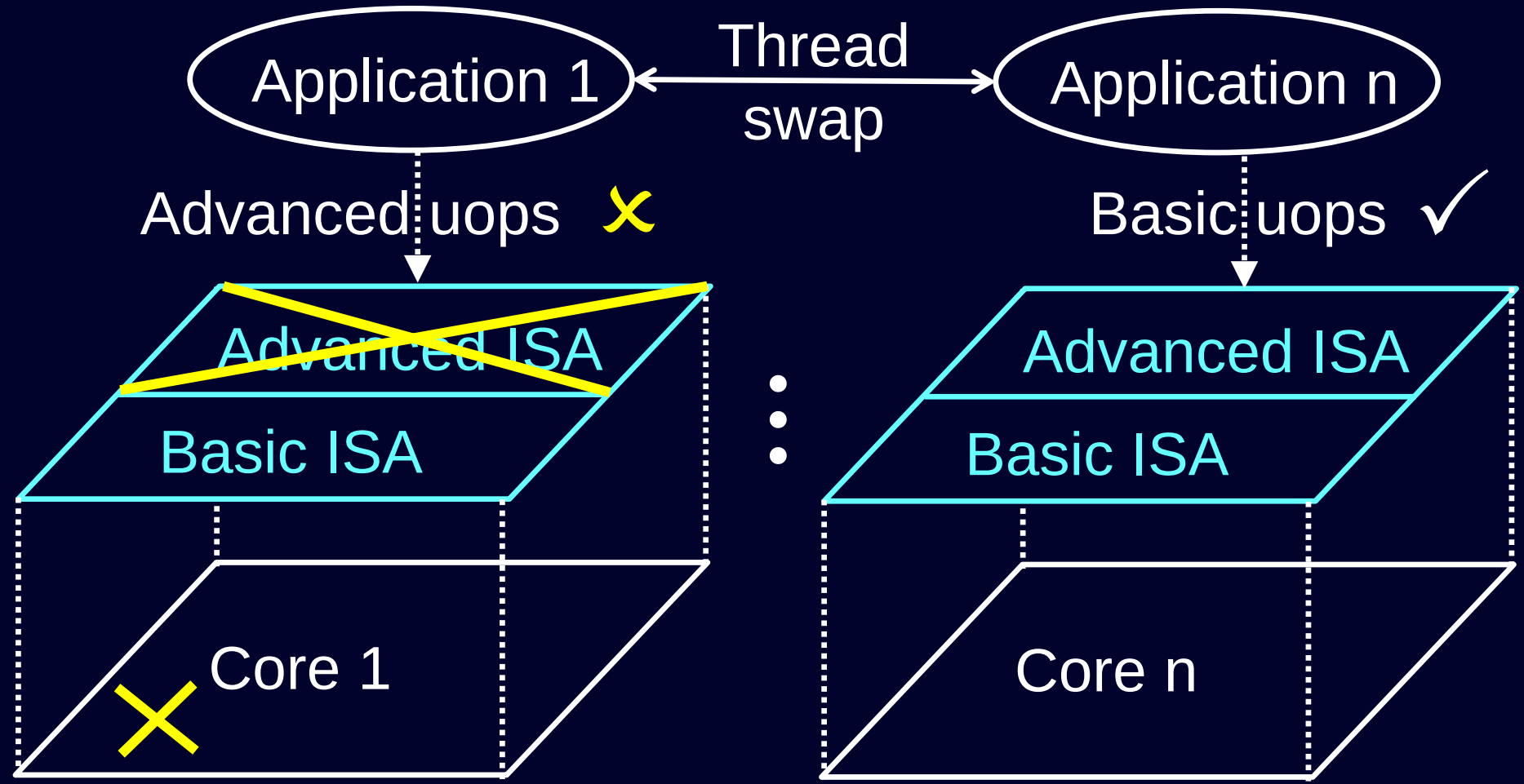
- Many-core CISC processor designs



[Powell 09]

Architectural Core Salvaging

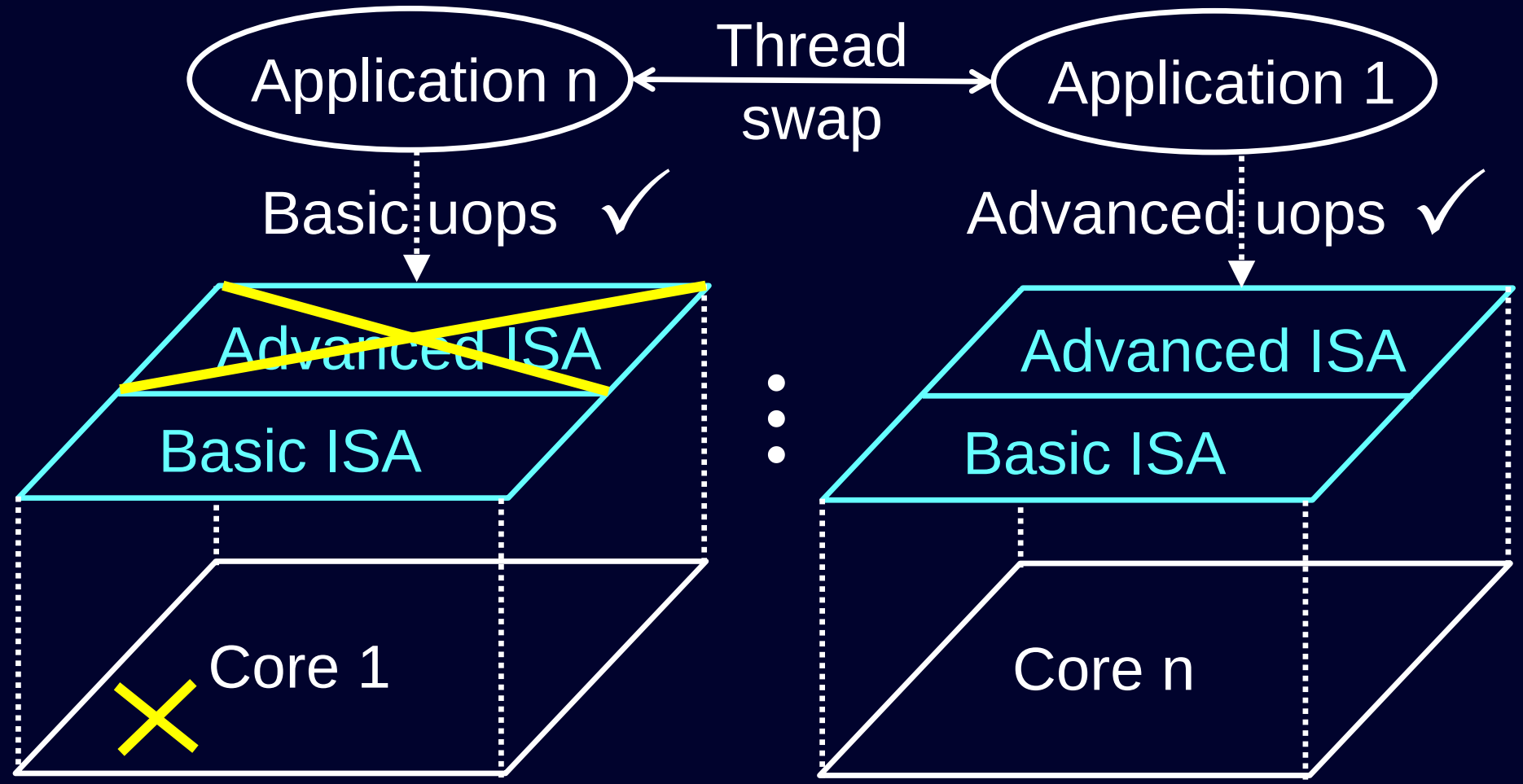
- Many-core CISC processor designs



[Powell 09]

Architectural Core Salvaging

- Many-core CISC processor designs



[Powell 09]

Architectural Core Salvaging: Discussion

- What about diagnosis
- Applicability
  - CISC-like architectures
- Performance
  - Depends
- Coverage issues
  - ~50% coverage of execution

Self-Repair for Processor Cores

- Which technique to choose?
- Software-assisted techniques?

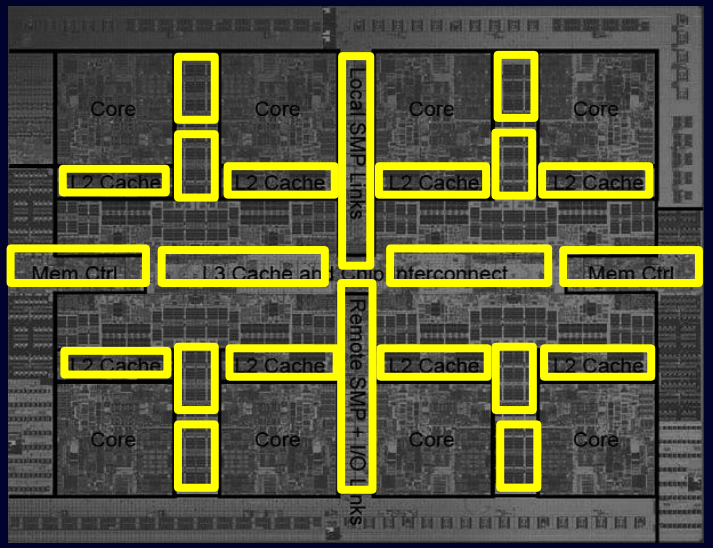
Outline

- Introduction
- Self-repair techniques
  - Memories
  - Processor cores
  - Uncore components
- Conclusion

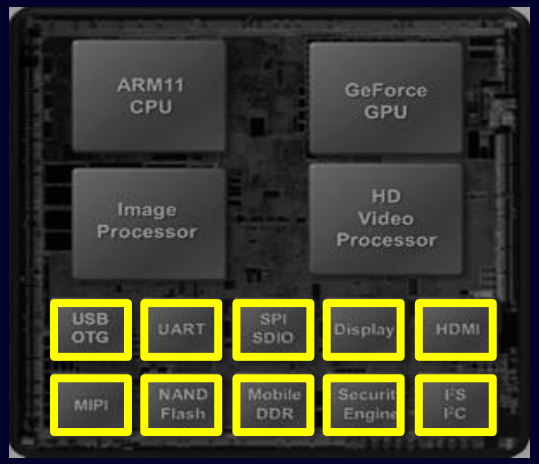
Uncore Prevalent in SoCs

- Uncore examples
  - Cache / DRAM controller
  - Accelerators
  - I/O interfaces

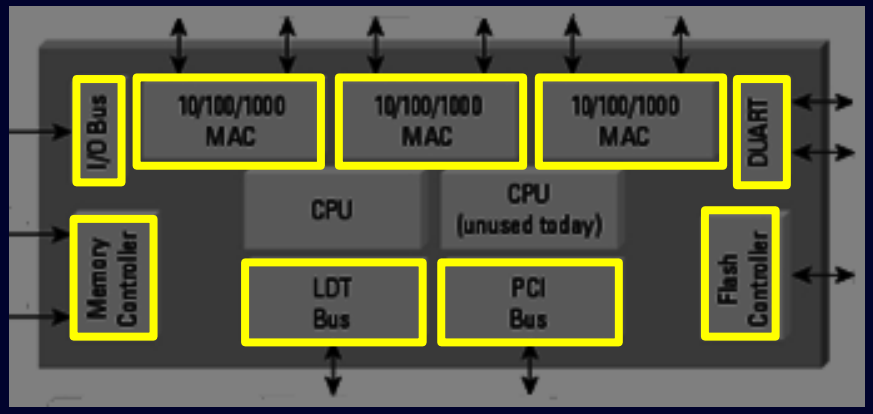
IBM Power 7



NVIDIA Tegra

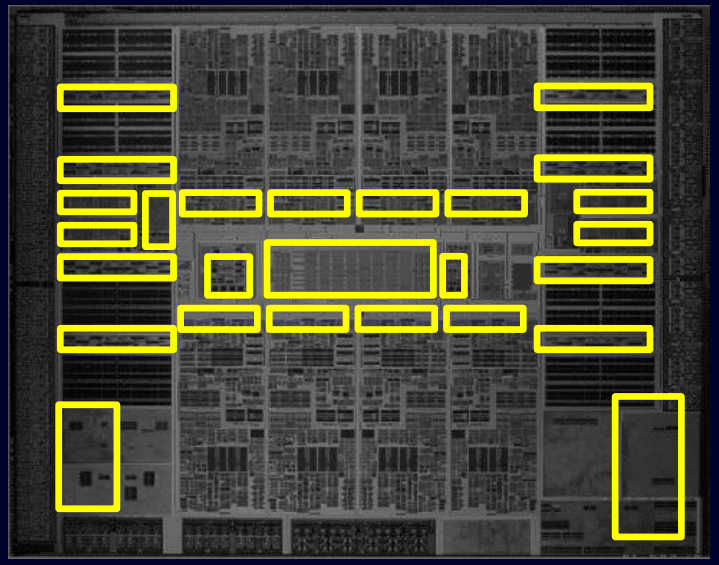


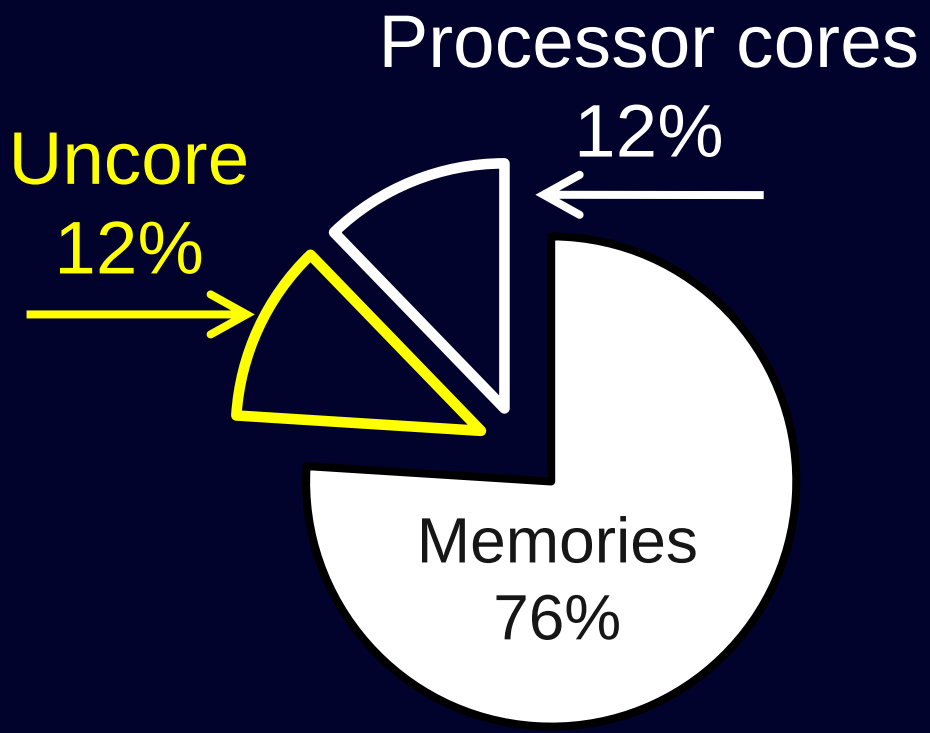
Cisco Network Processor



Uncore Self-Repair Essential

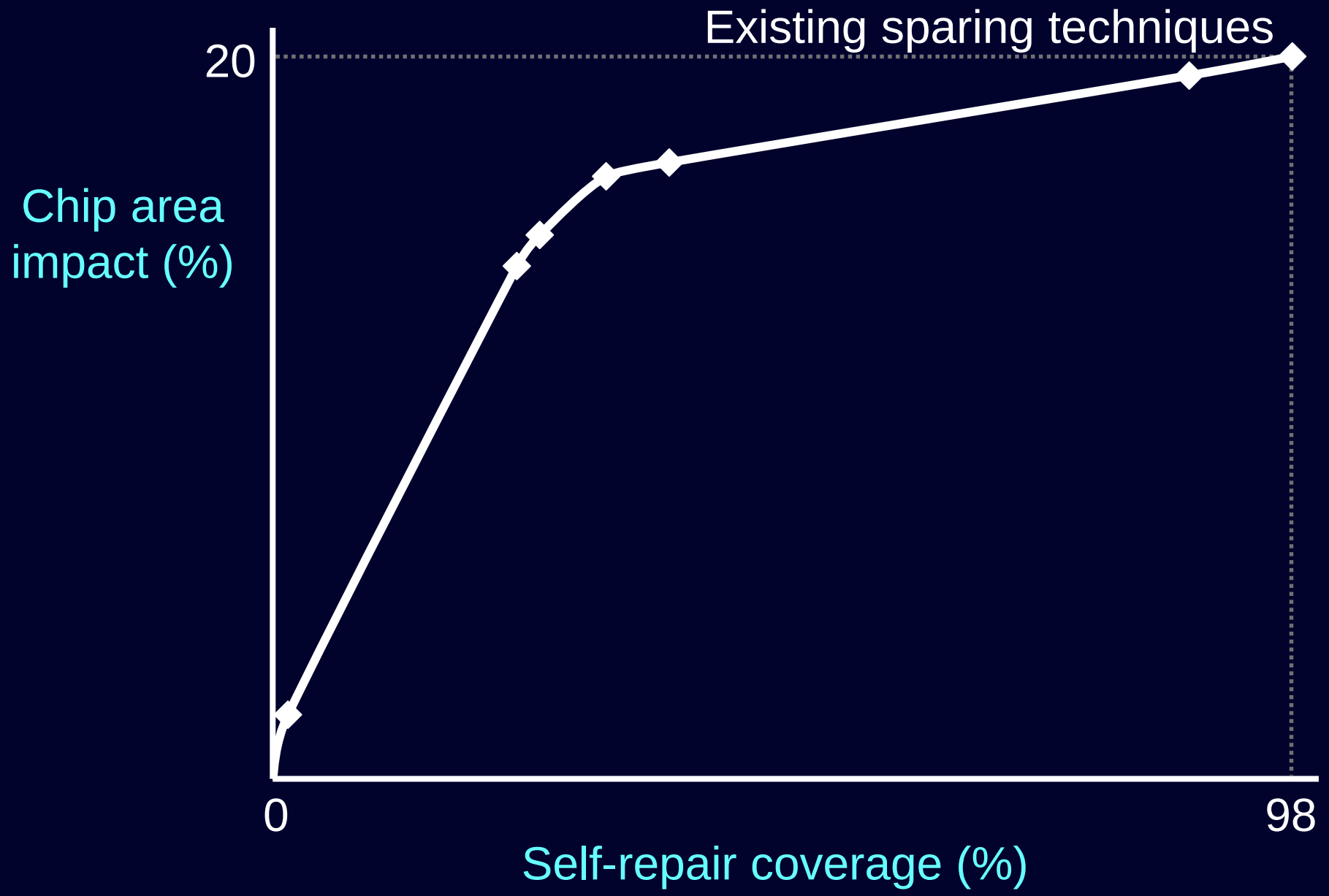
OpenSPARC T2



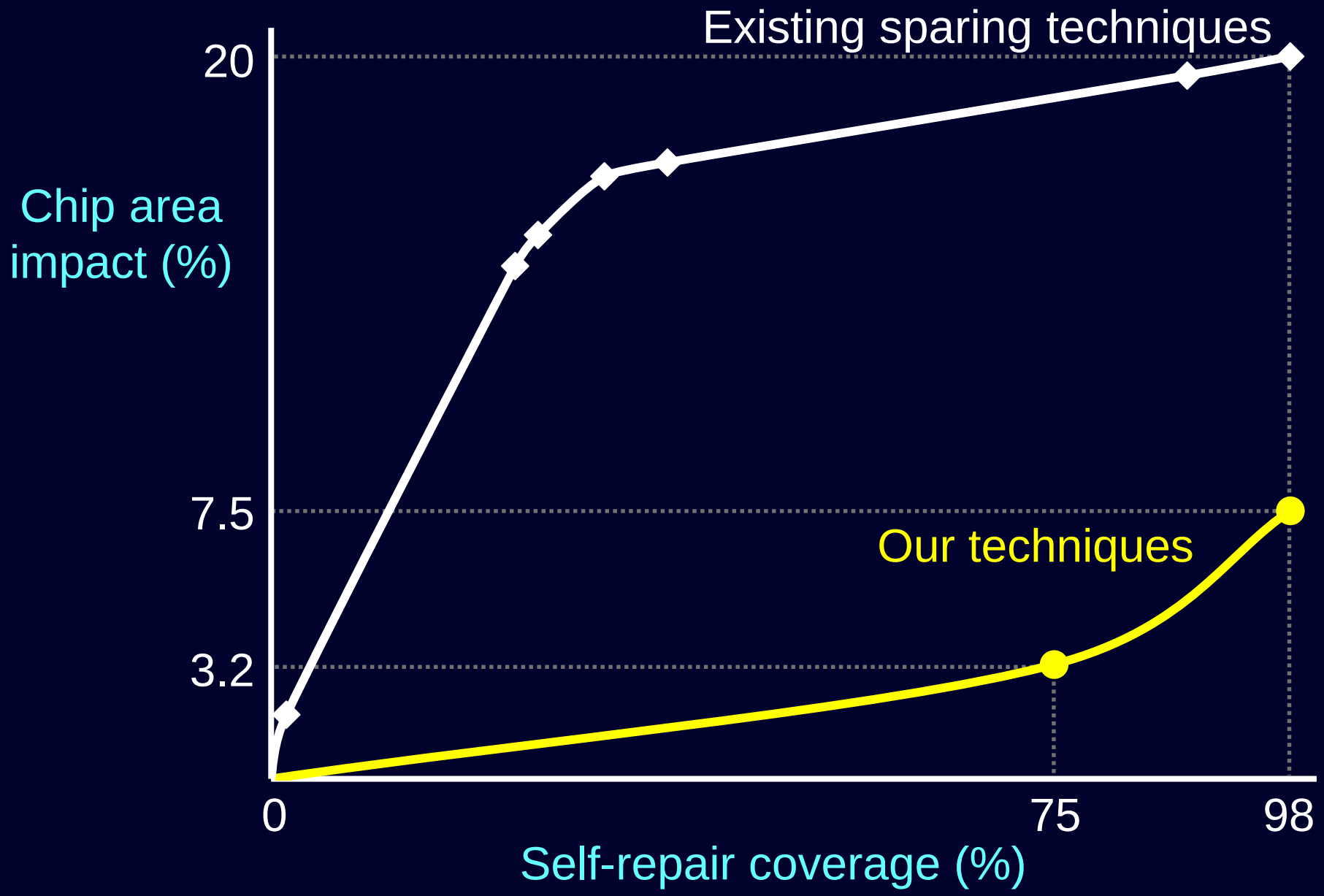


- Cores, memories, networks-on-chip
  - Many existing techniques

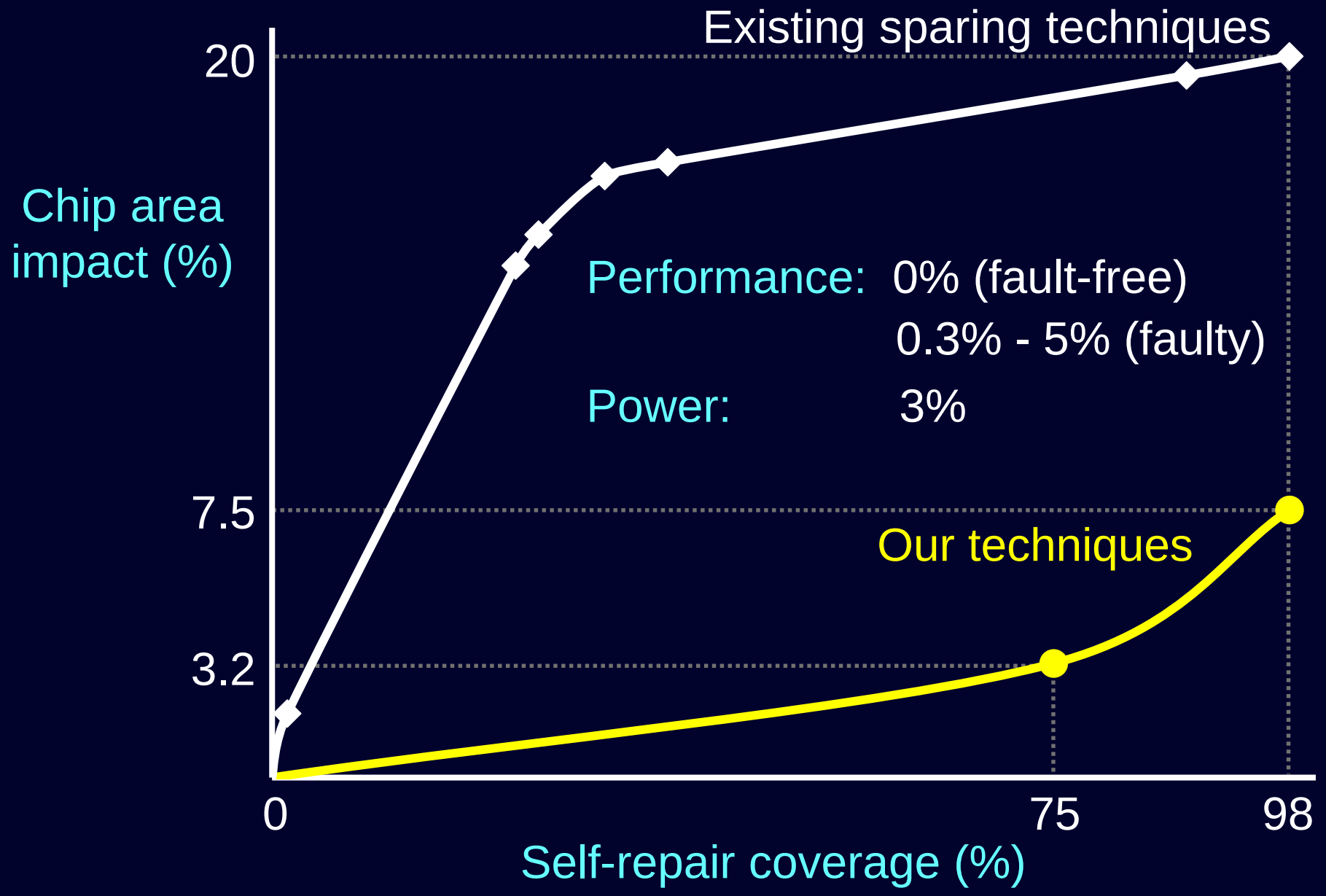
Key Message [Li ITC13]



Key Message [Li ITC13]

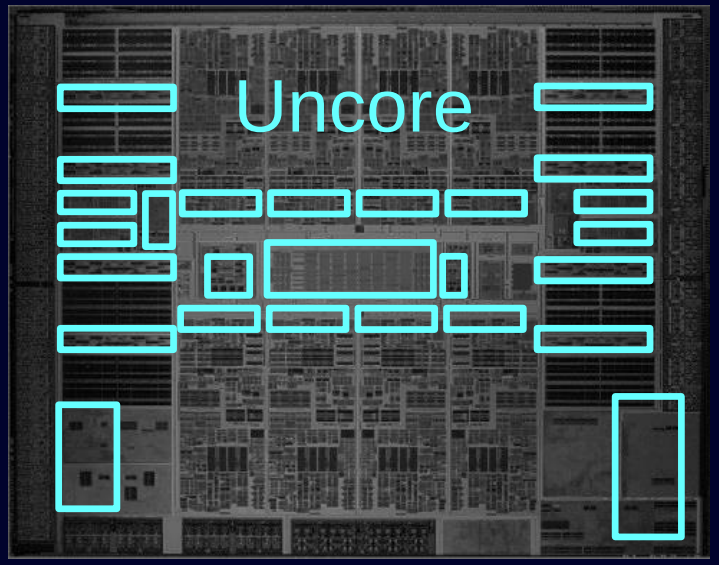


Key Message [Li ITC13]



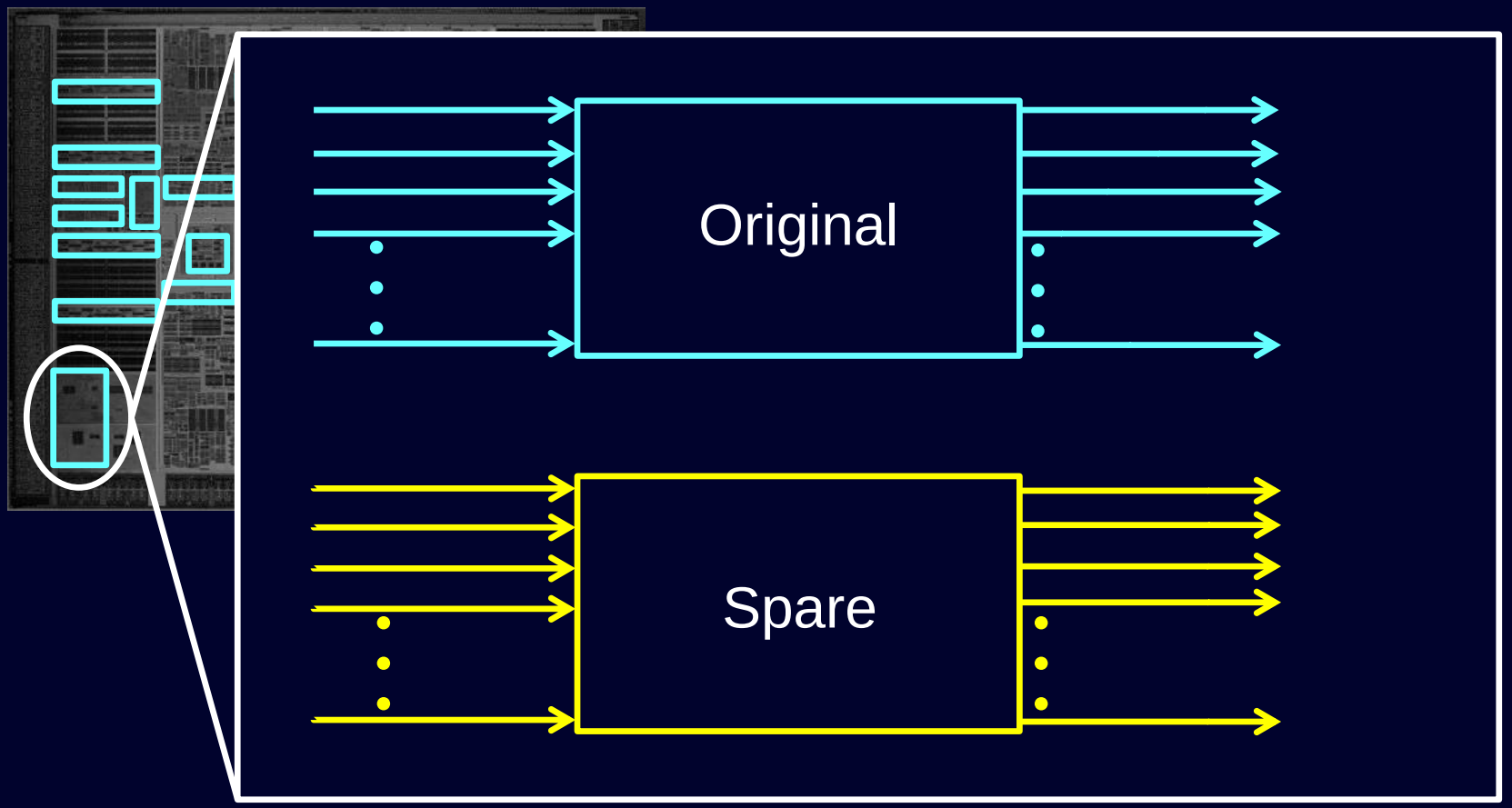
Existing Techniques Inadequate

OpenSPARC T2

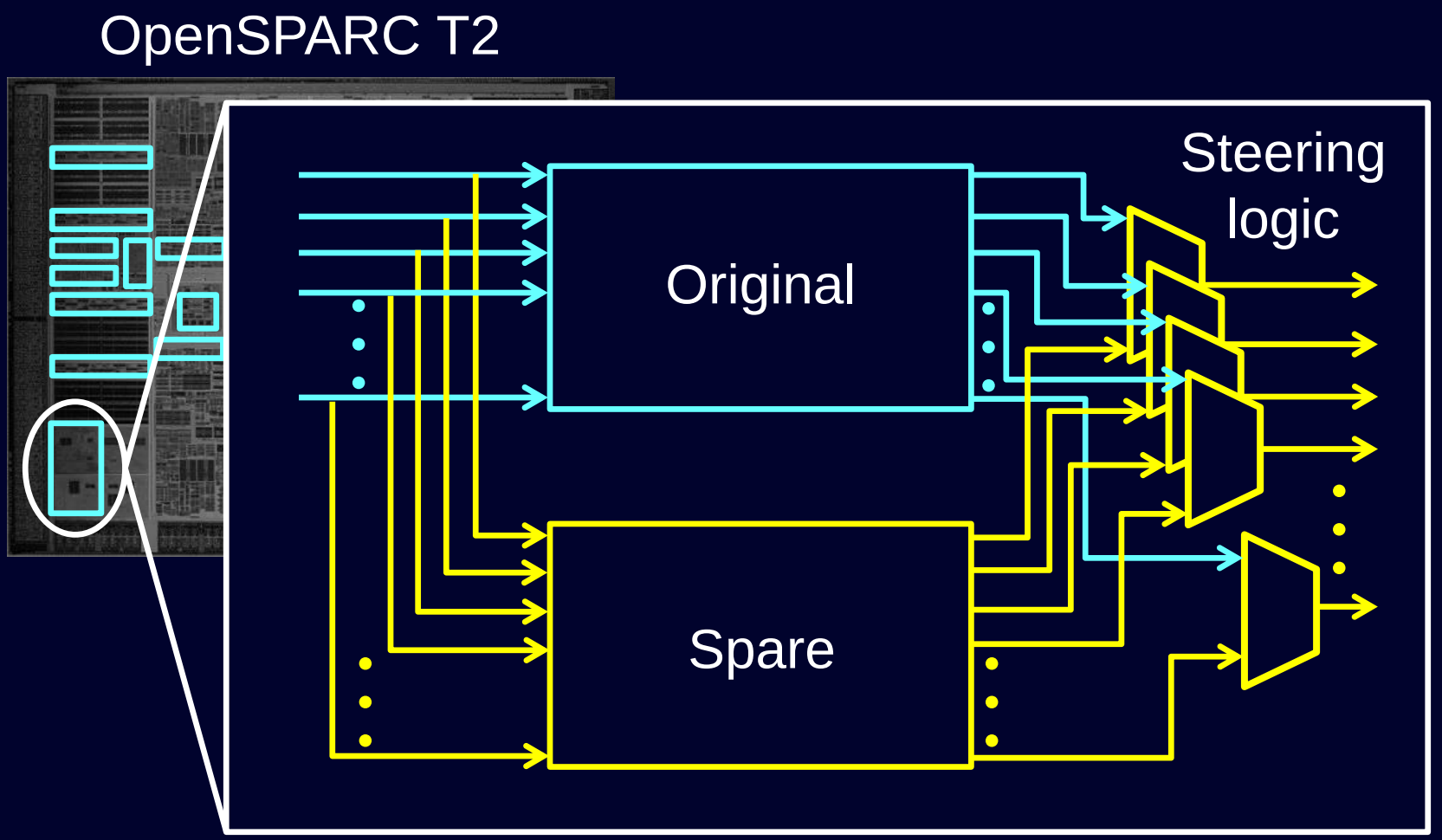


Existing Techniques Inadequate

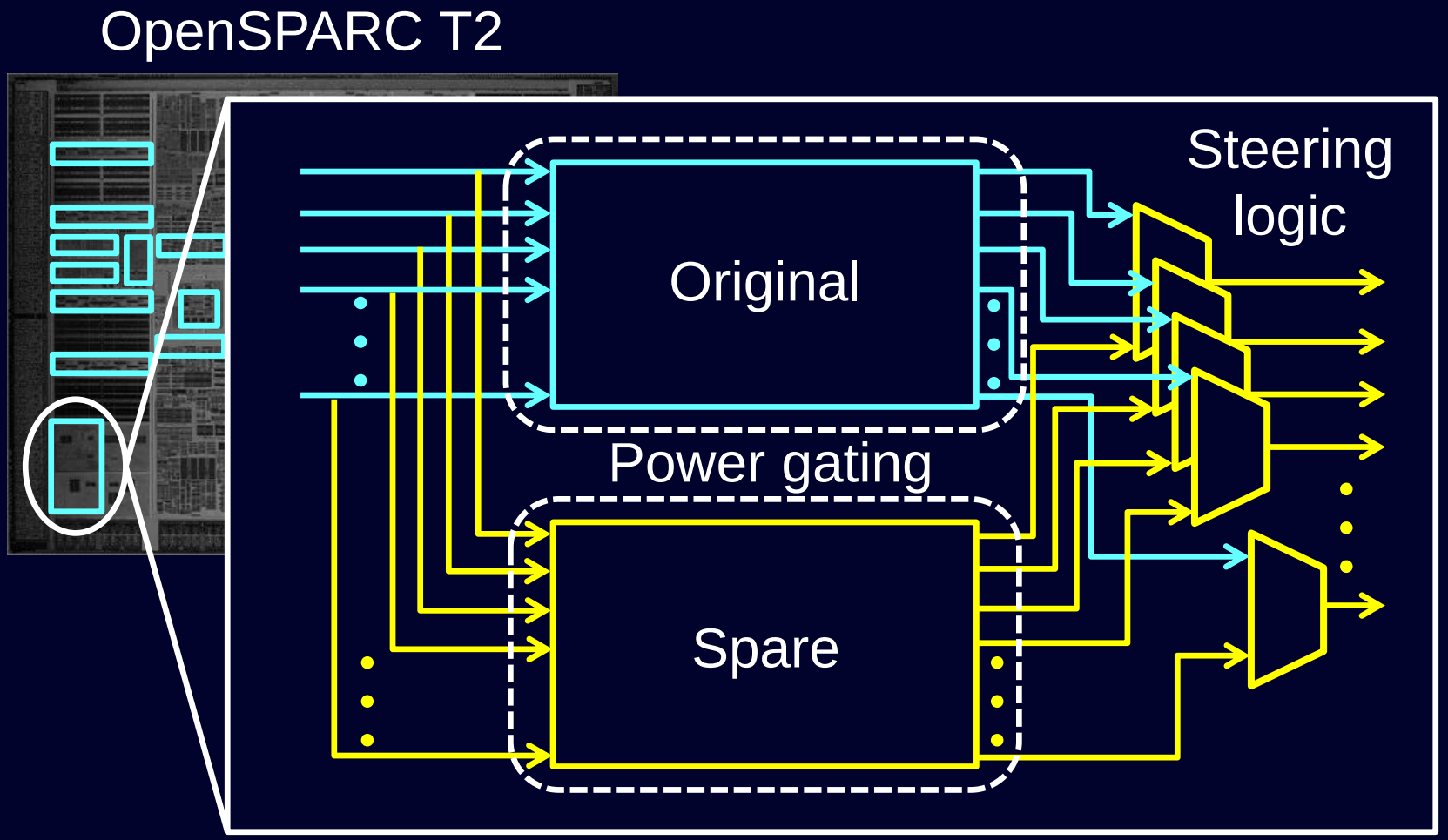
OpenSPARC T2



Existing Techniques Inadequate



Existing Techniques Inadequate



☹ 20% chip area cost

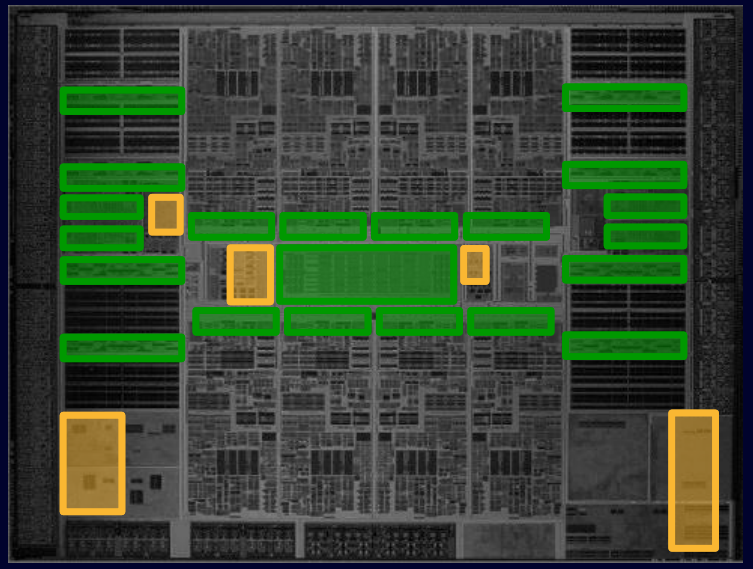
New Uncore Self-Repair Techniques

- ERRS: Enhanced Resource Reallocation and Sharing
- SHE: Sparing through Hierarchical Exploration

OpenSPARC T2

8 cores, 64 threads

500M transistors



☺ 7.5% chip area cost

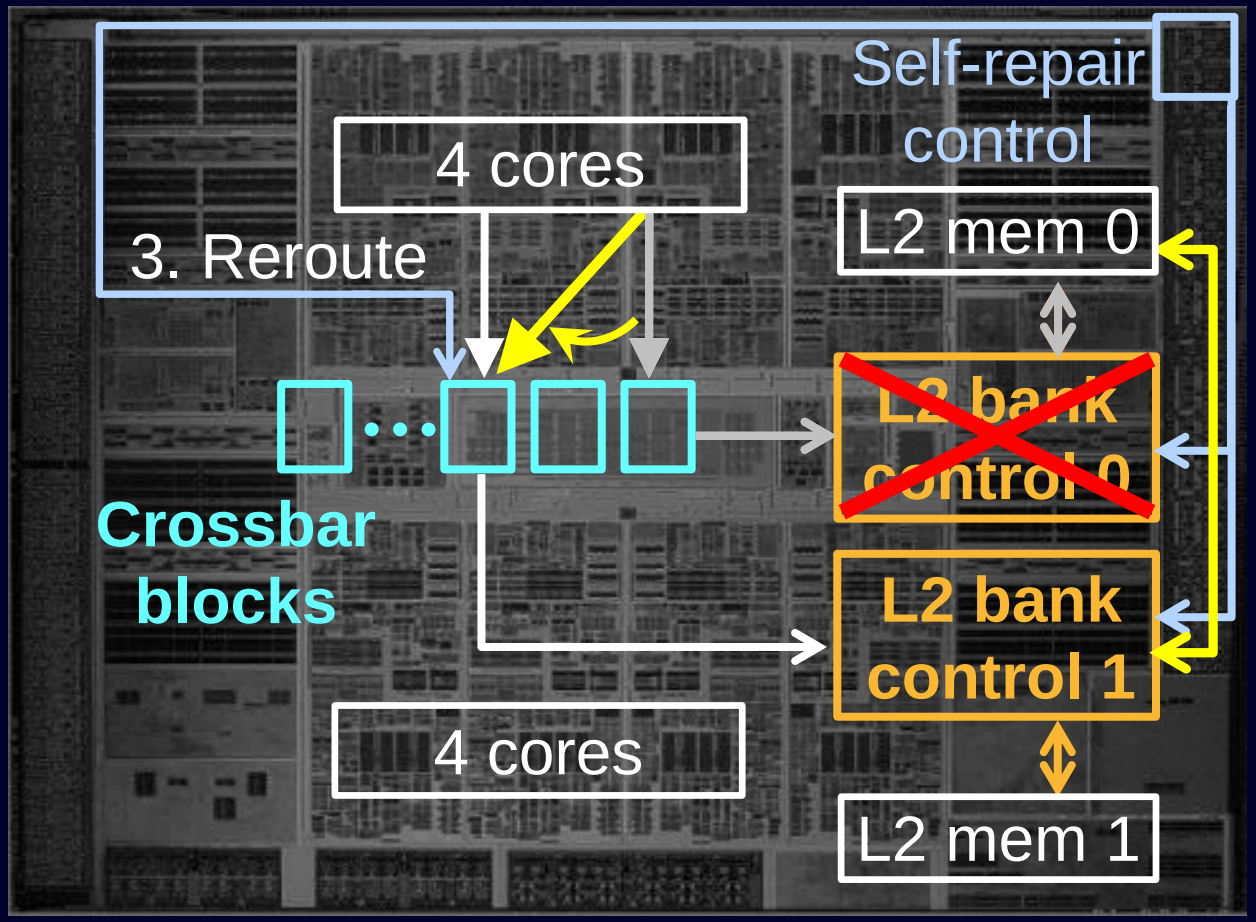
Outline

- Introduction
- Self-repair for uncore
  - ERRS
  - SHE
- Conclusion

Basic Resource Reallocation & Sharing (RRS)

- Already-existing "similar" components

☺ No spares



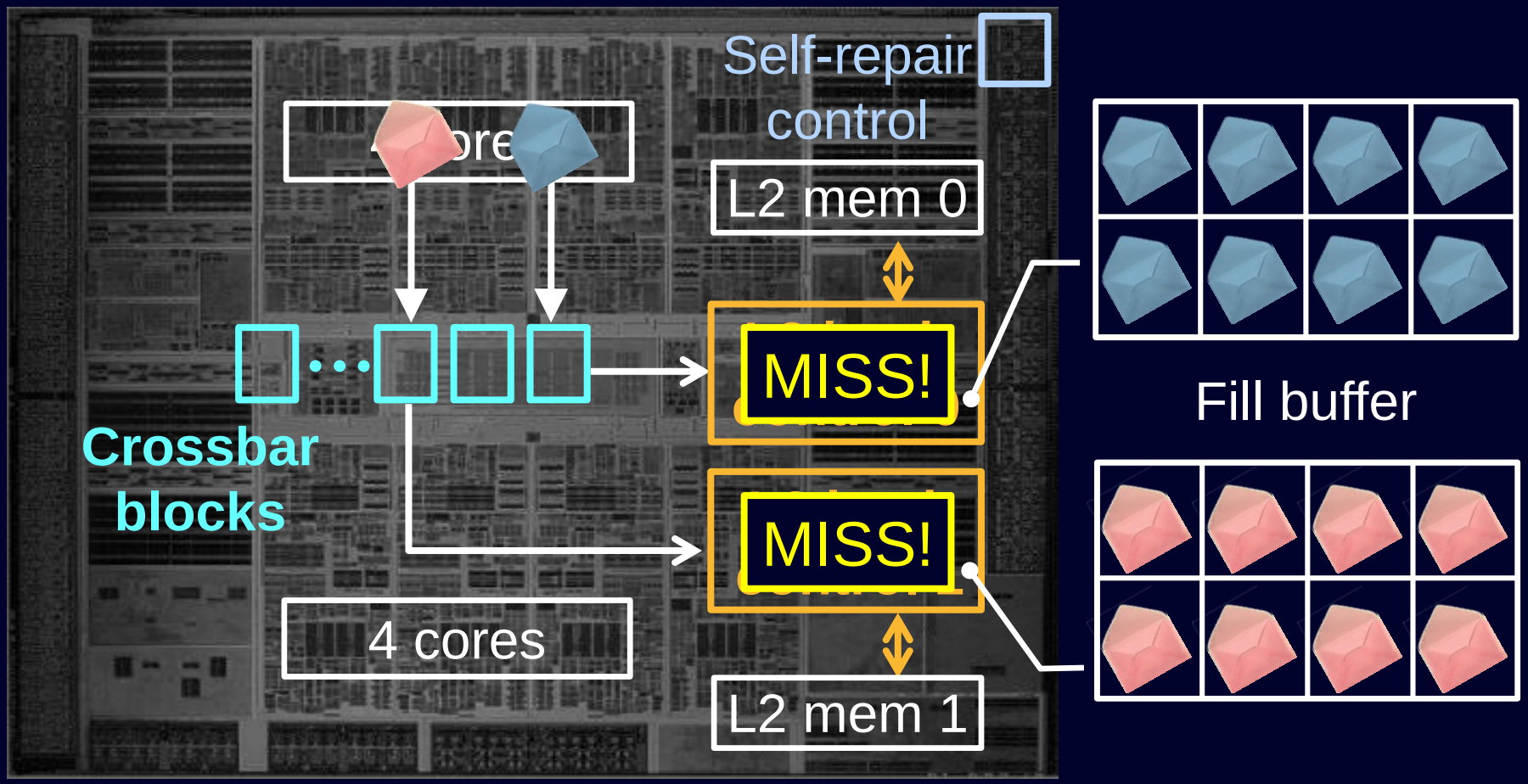
1. Fault detected

8 cycle overhead (faulty case only)

2. Resource sharing

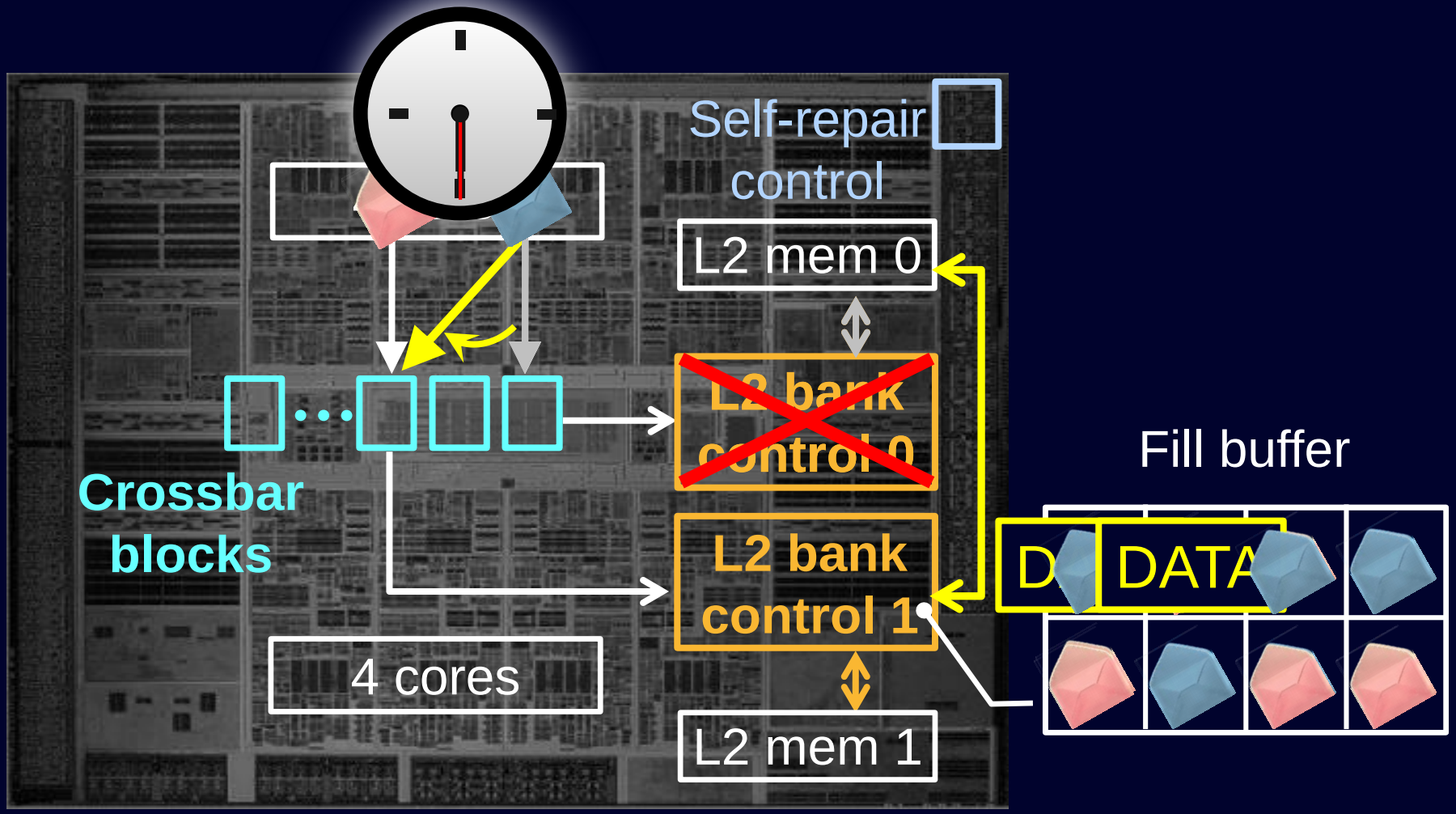
Basic RRS Performance Impact

Fault-free scenario

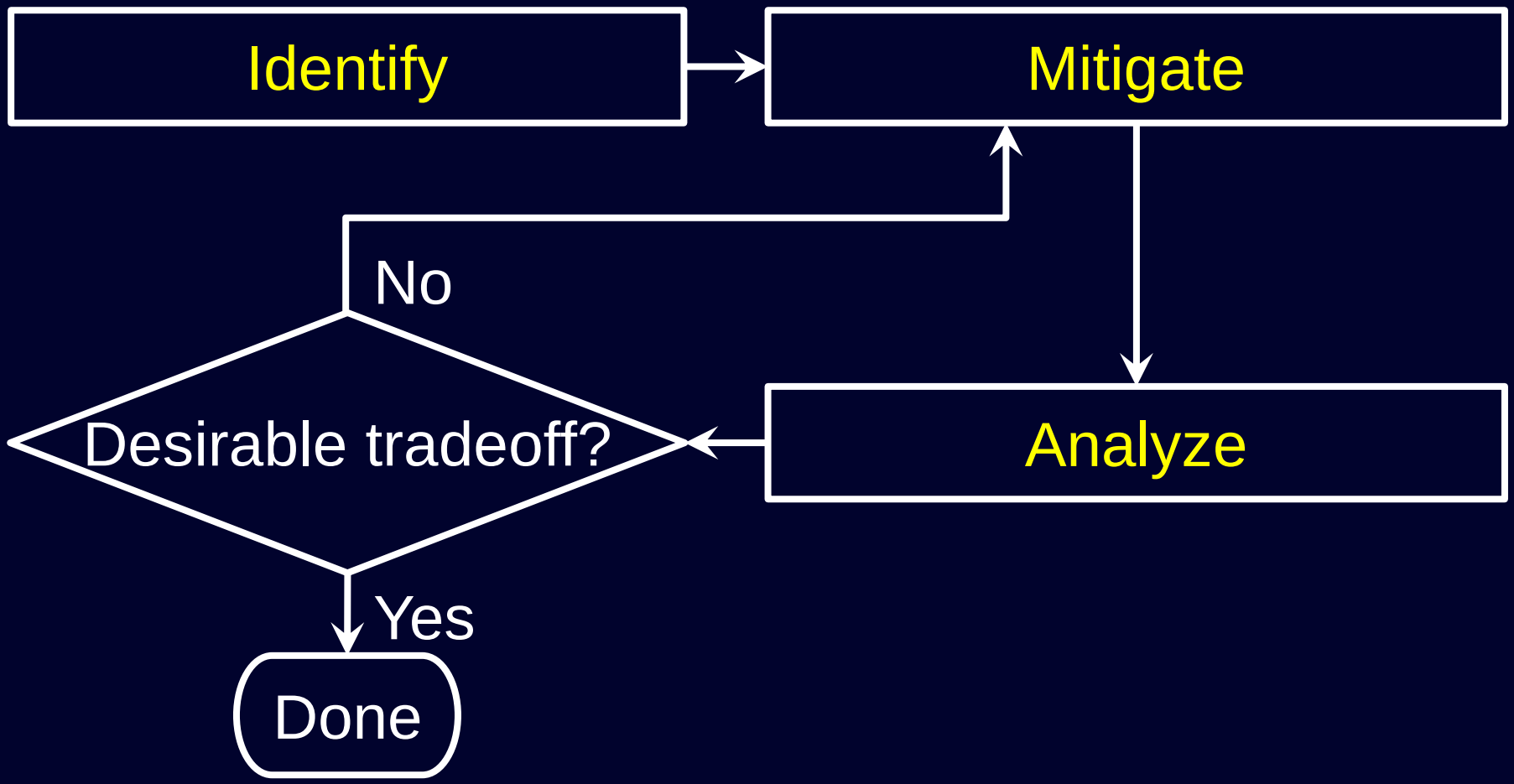


Basic RRS Performance Impact

Single faulty component: 70% impact

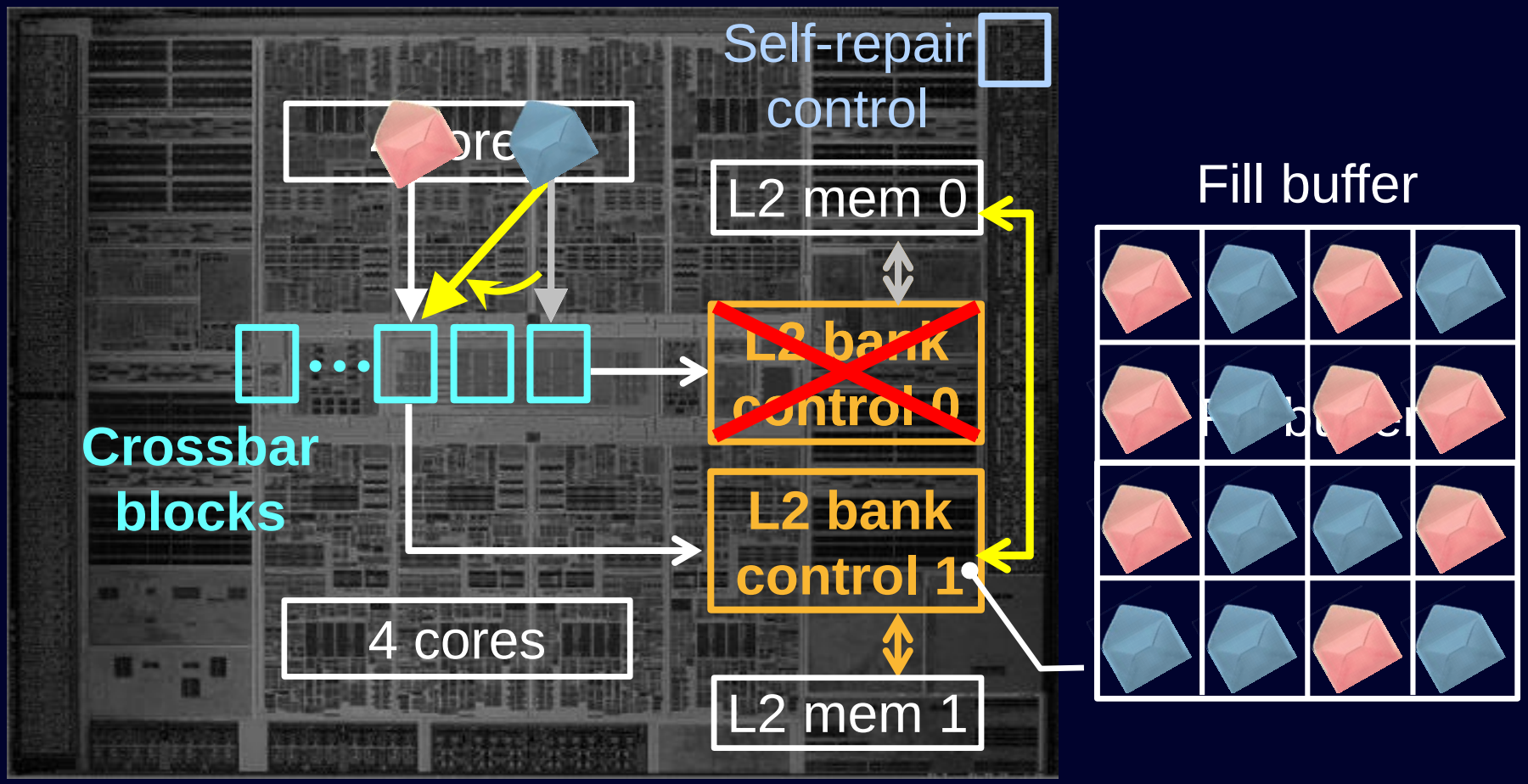


Enhanced RRS (ERRS) Idea

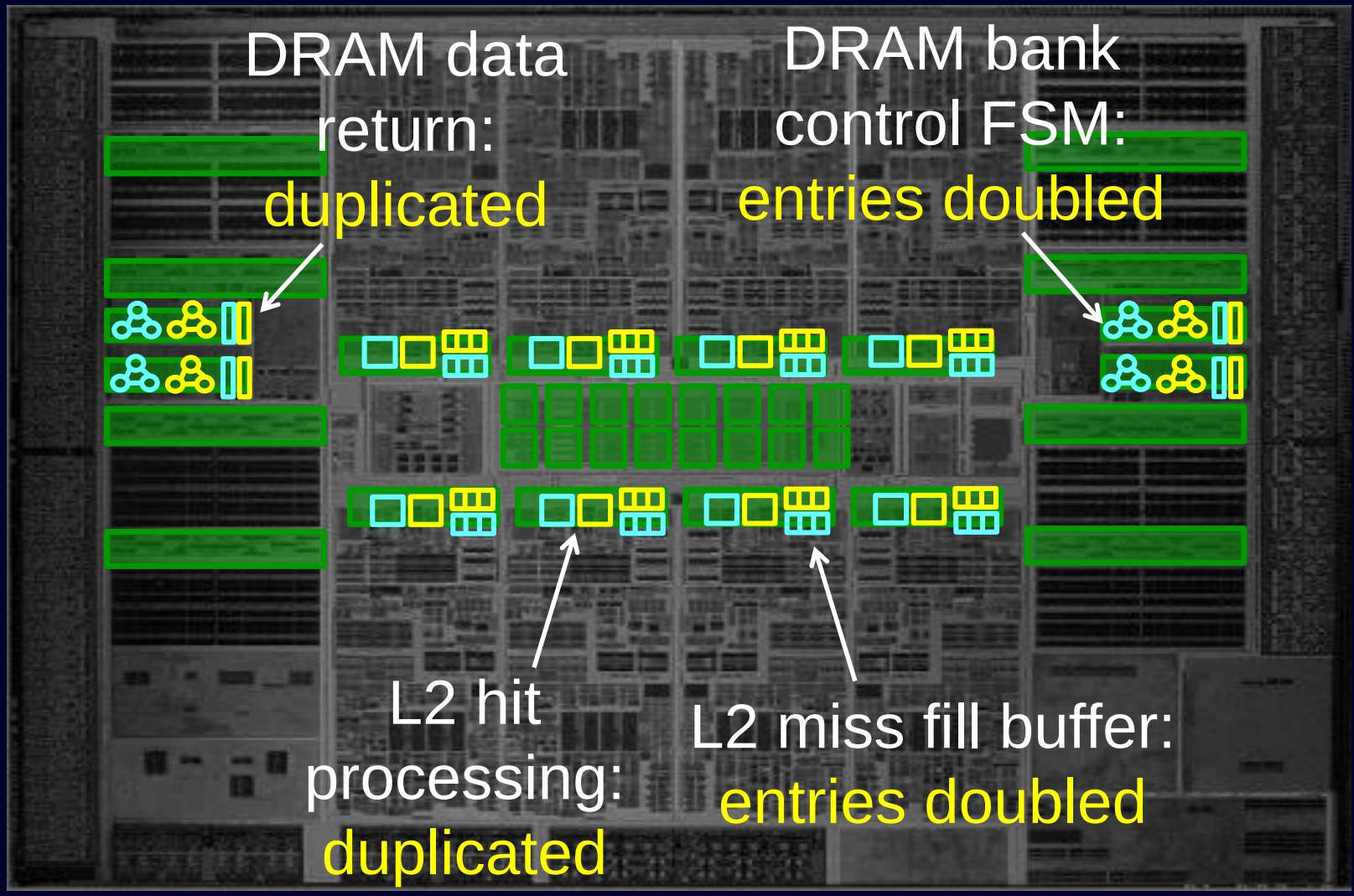


ERRS Mitigates RRS Performance Impact

Single faulty component: 3% impact



ERRS on OpenSPARC T2



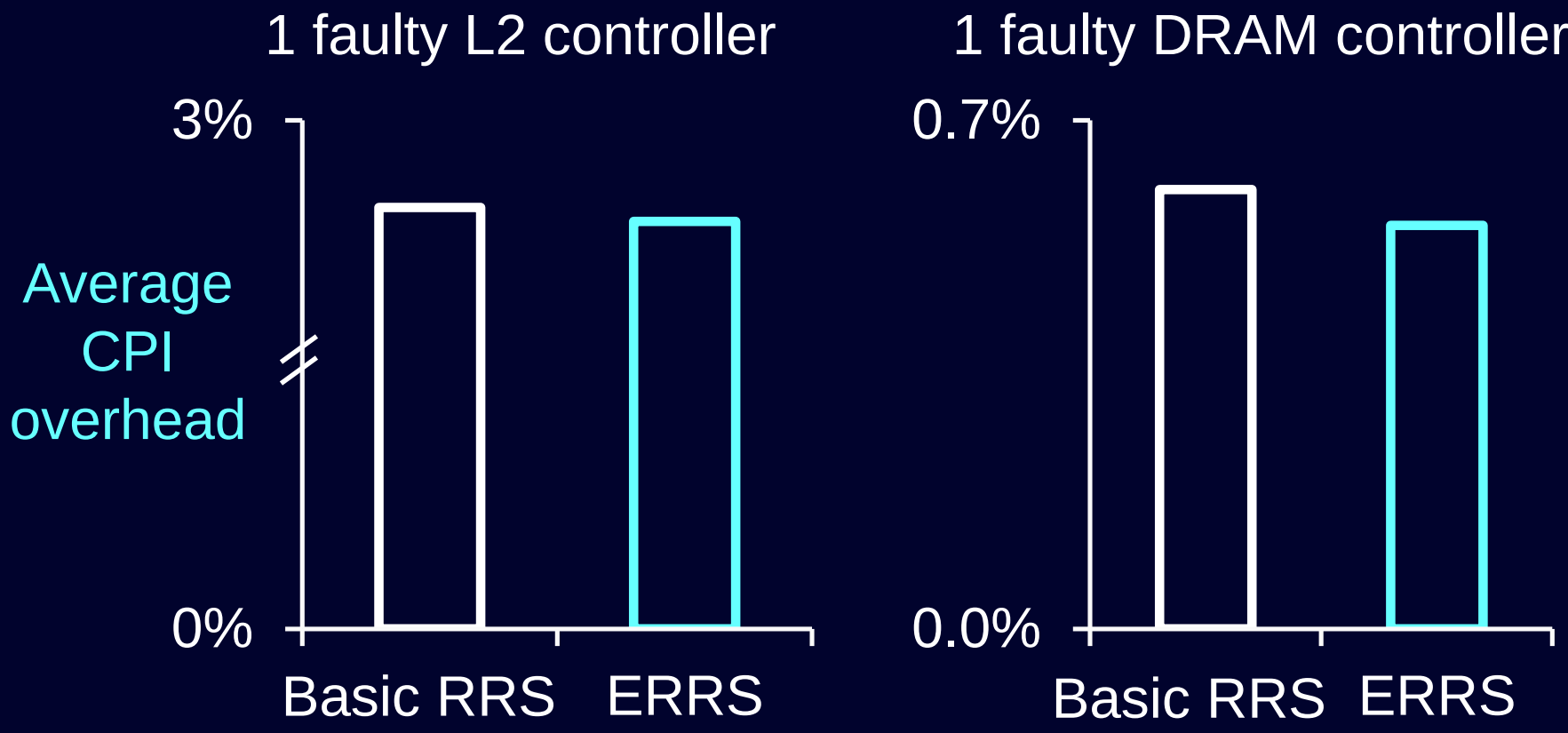
☺ 3.2% area, 2.7% power

Performance Evaluation Setup

Simulators	GEM5
Simulated CMP	64 single-issue in-order processor cores 1KB private L1 data and instruction caches 4MByte shared L2 cache (8 banks) 4 DRAM controllers

ERRS vs. Basic RRS: Performance Impact

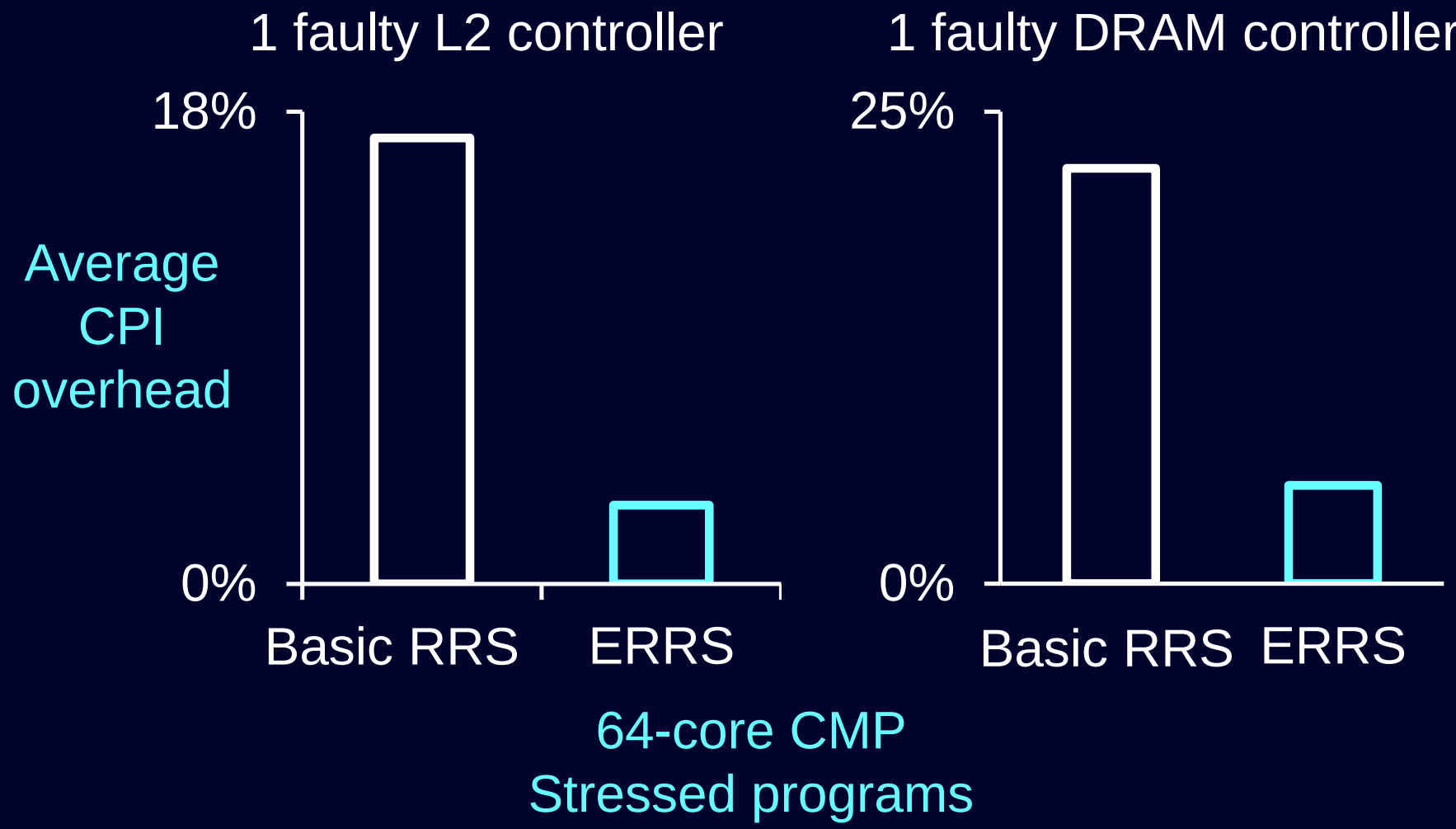
ERRS: 3% performance impact



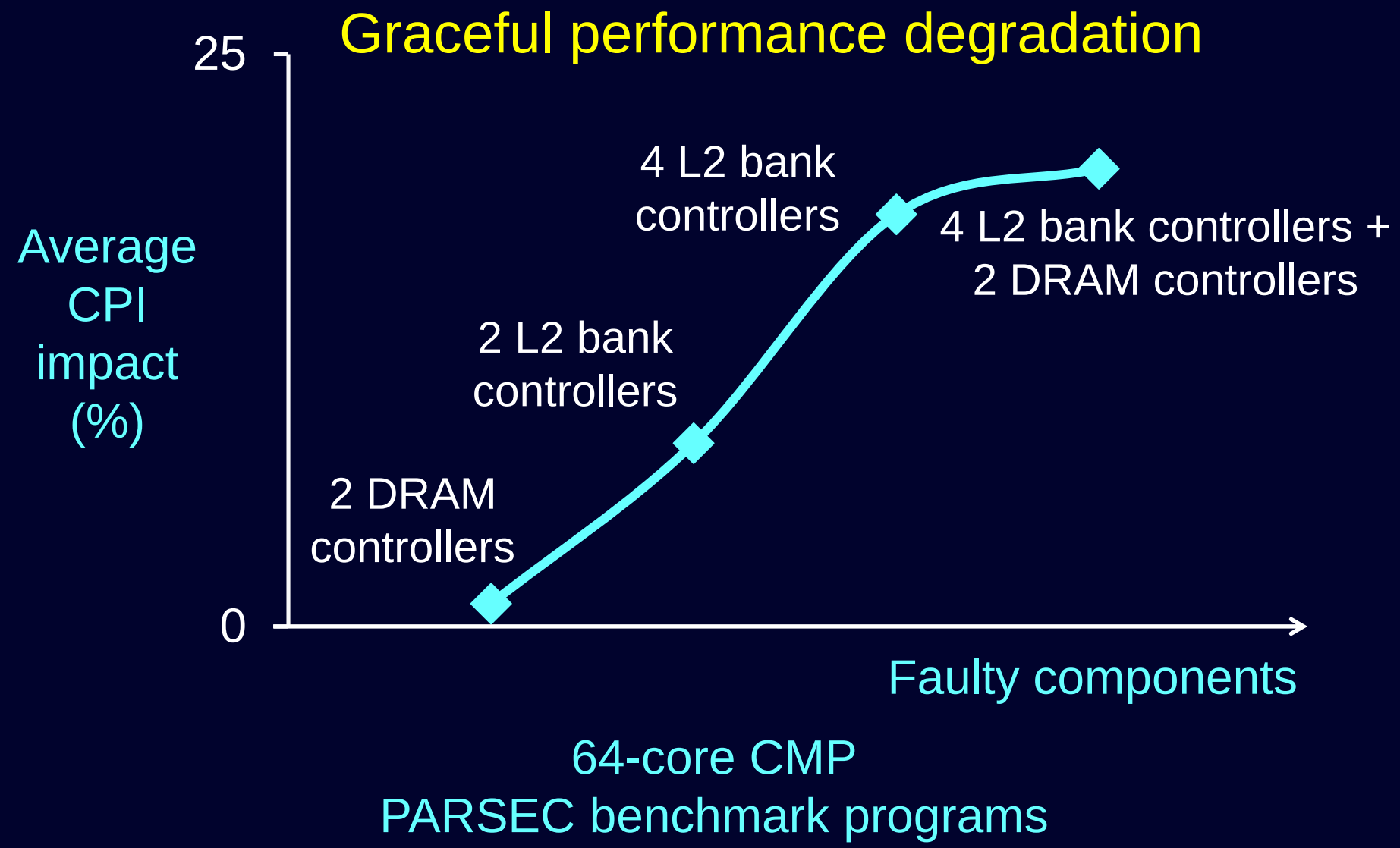
64-core CMP
PARSEC benchmark programs

ERRS vs. Basic RRS: Performance Impact

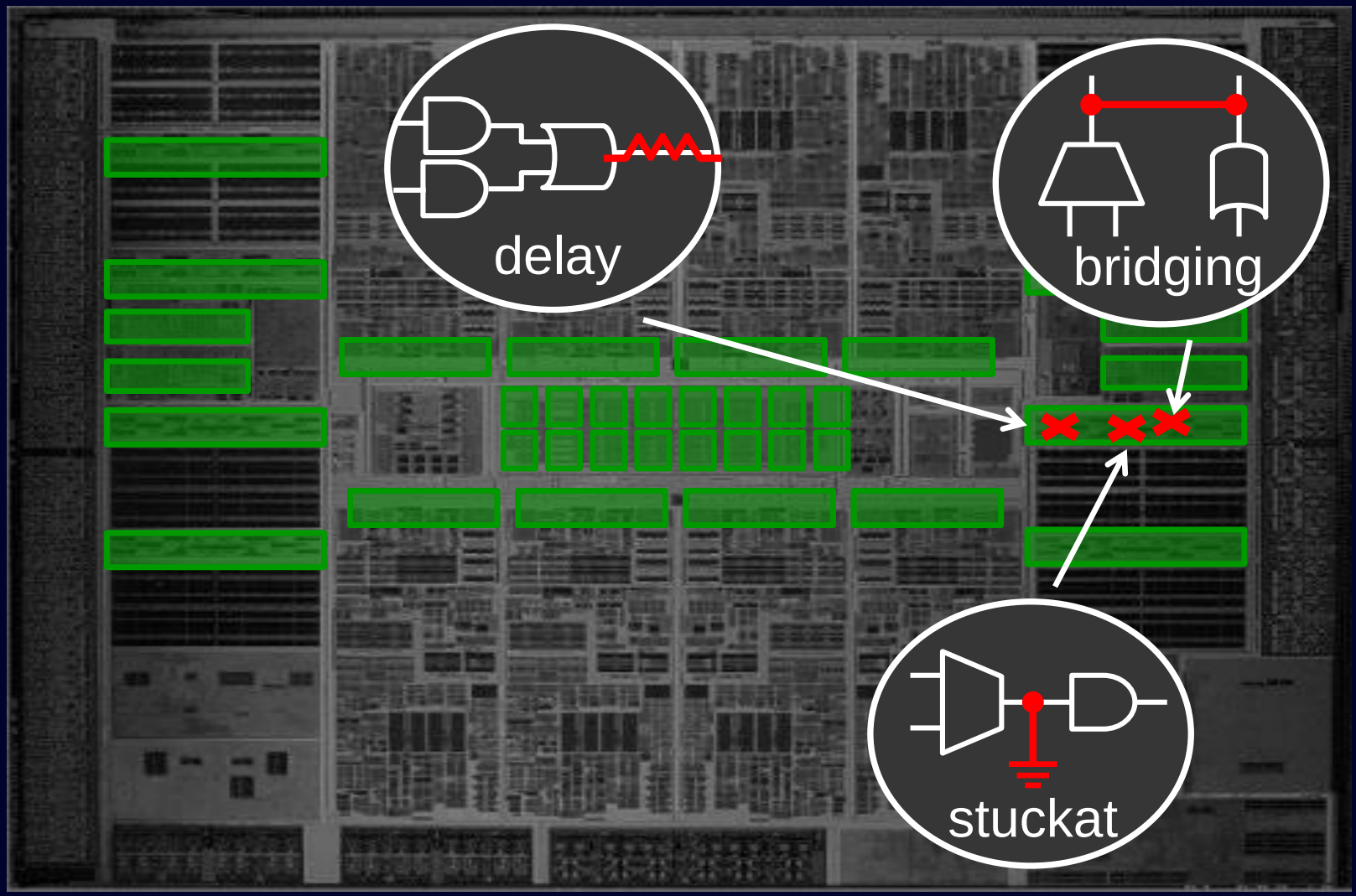
ERRS: 5% performance impact



ERRS for Multiple Faulty Components



Which Faults Repairable?

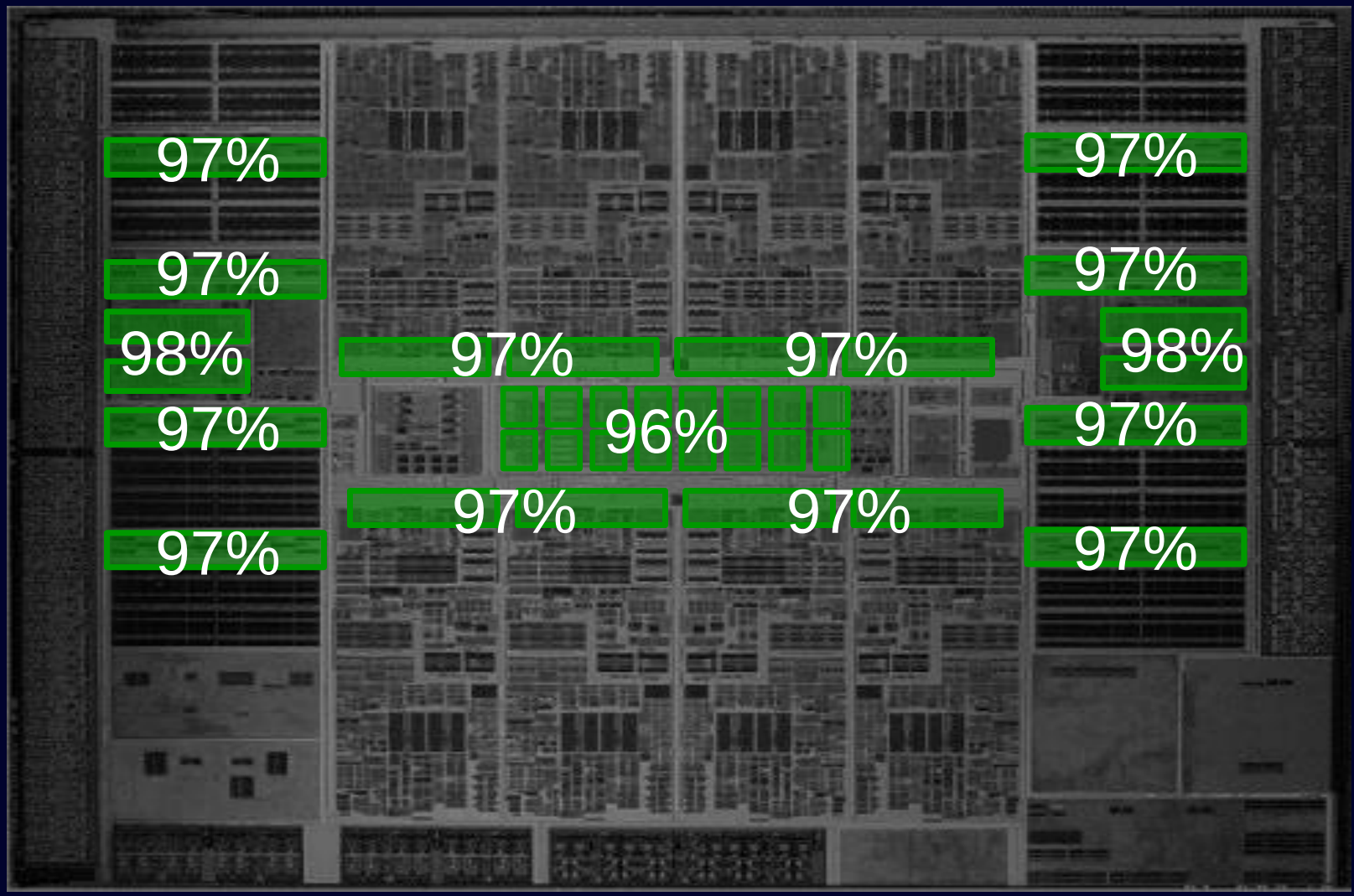


☺ All faults inside a component

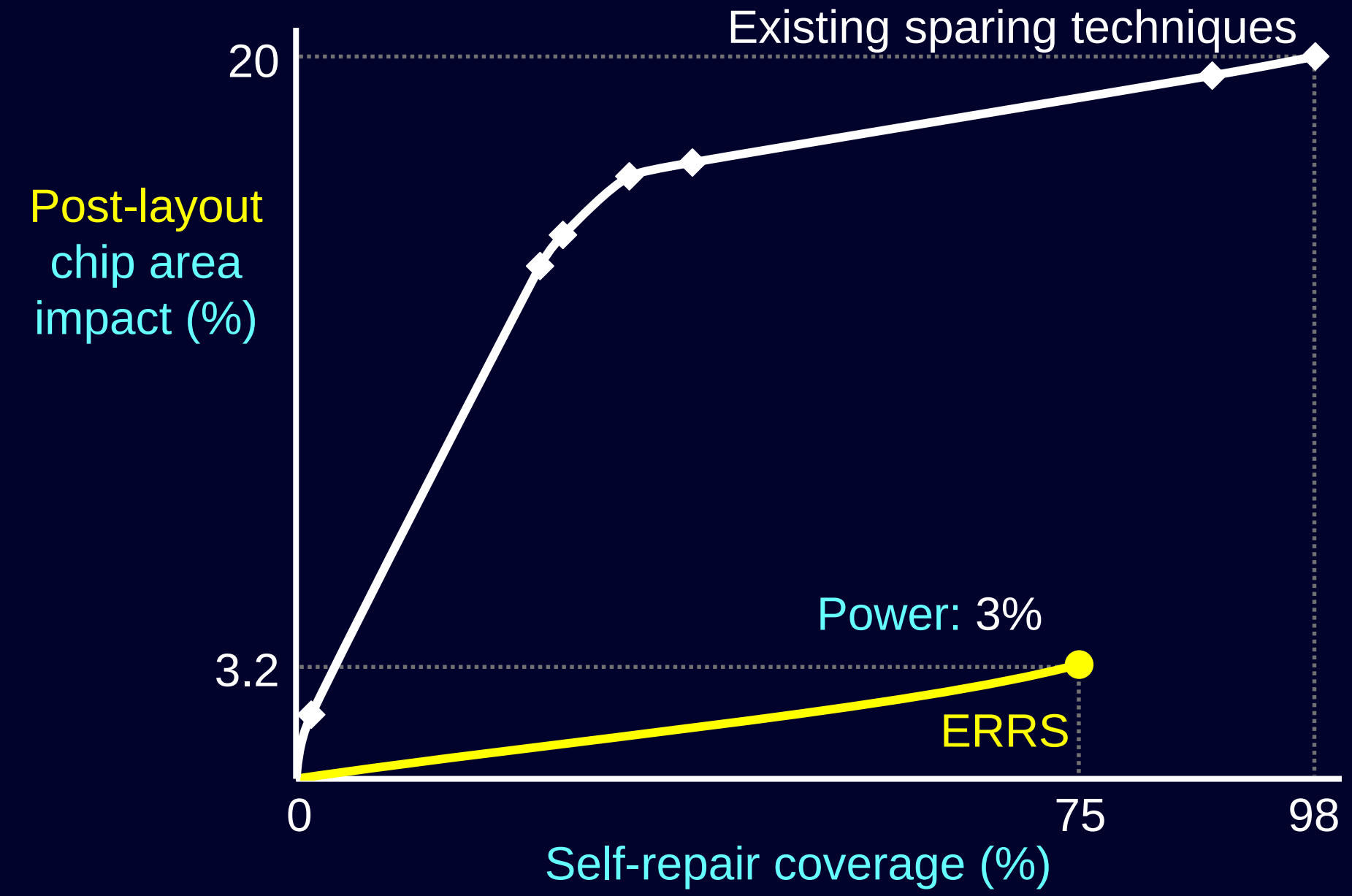
Which Faults Not Repairable?

- Single points of failure
  - Primary inputs
  - Primary outputs
  - Steering logic
- Metric: *self-repair coverage*
  - % non-single points of failure

ERRS Component Self-Repair Coverage



ERRS Chip Area/Power Impact & Coverage



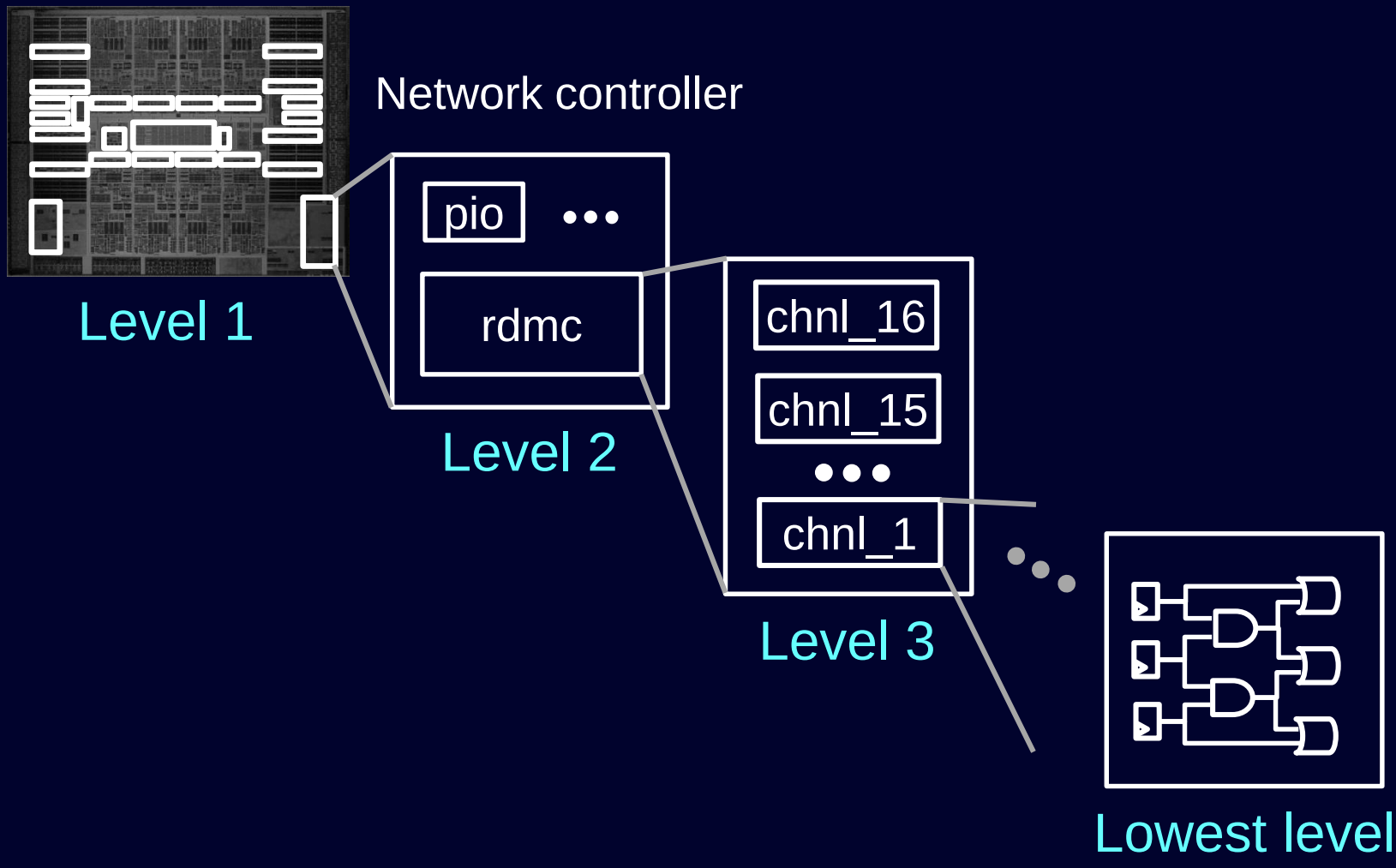
Outline

- Introduction
- Self-repair for uncore
  - ERRS
  - SHE
- Conclusion

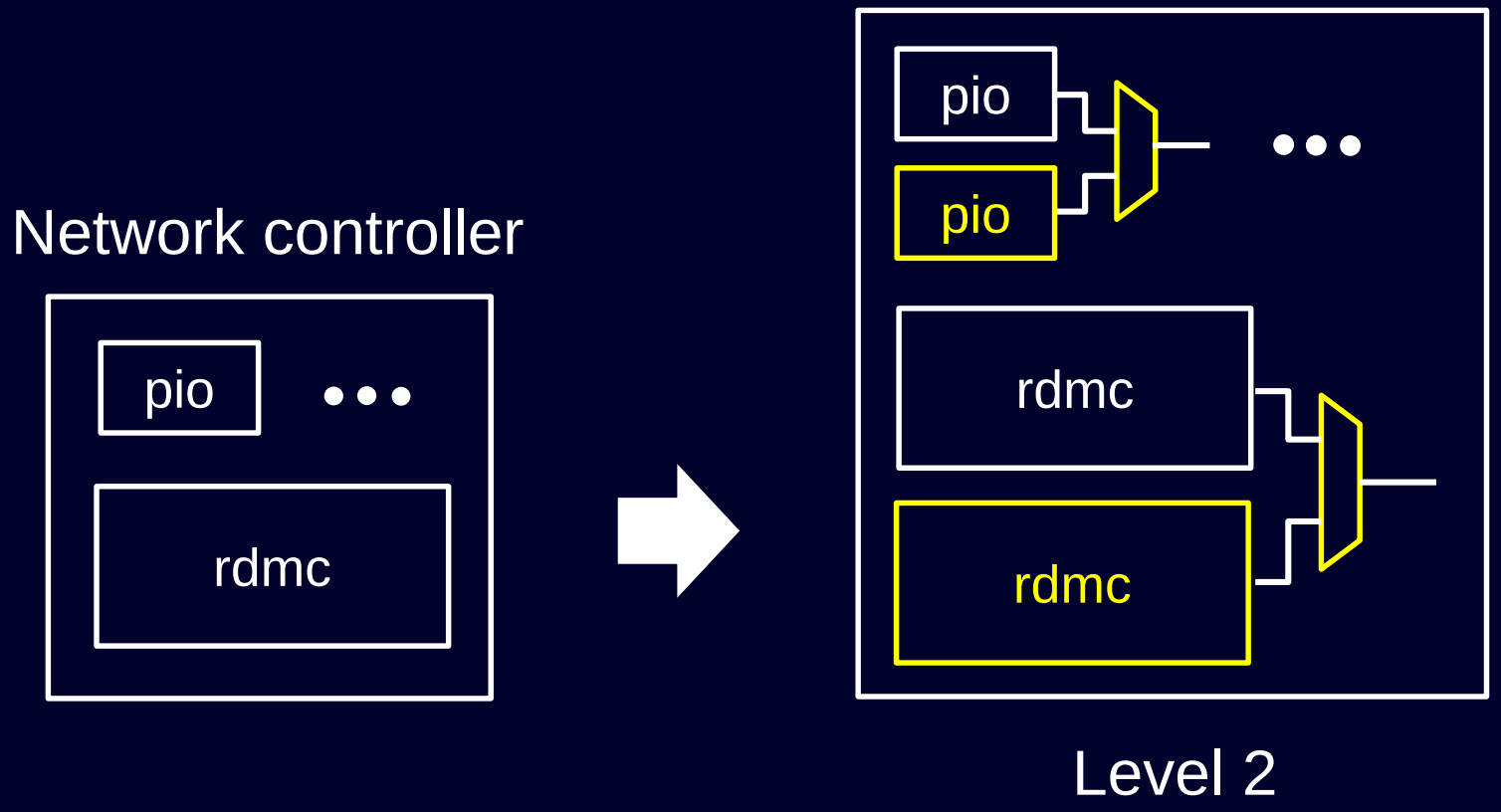
SHE: Sparing through Hierarchical Exploration

- Minimize area cost
  - Identical blocks → spare shared
- Balanced coverage vs. area

The Design Hierarchy

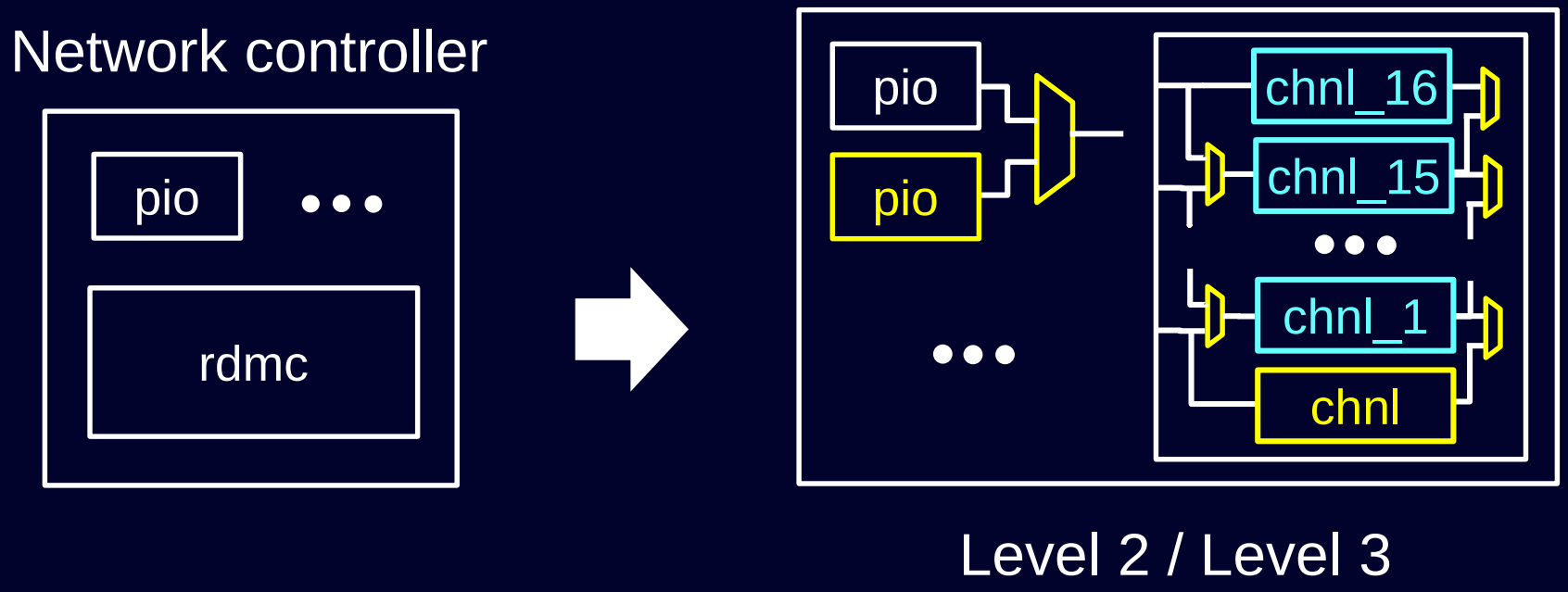


Sparing in Different Levels: Coarse-Grained

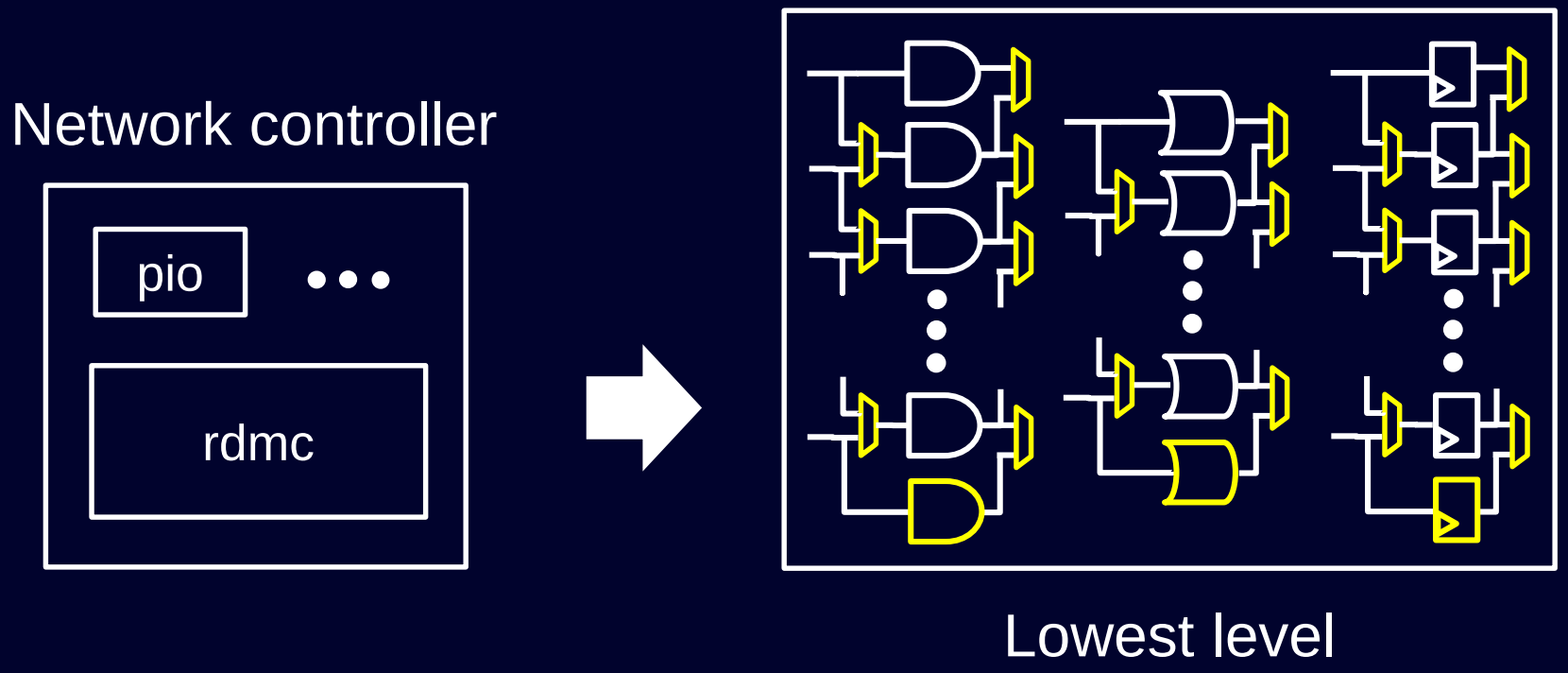


Original
Spare / steering logic

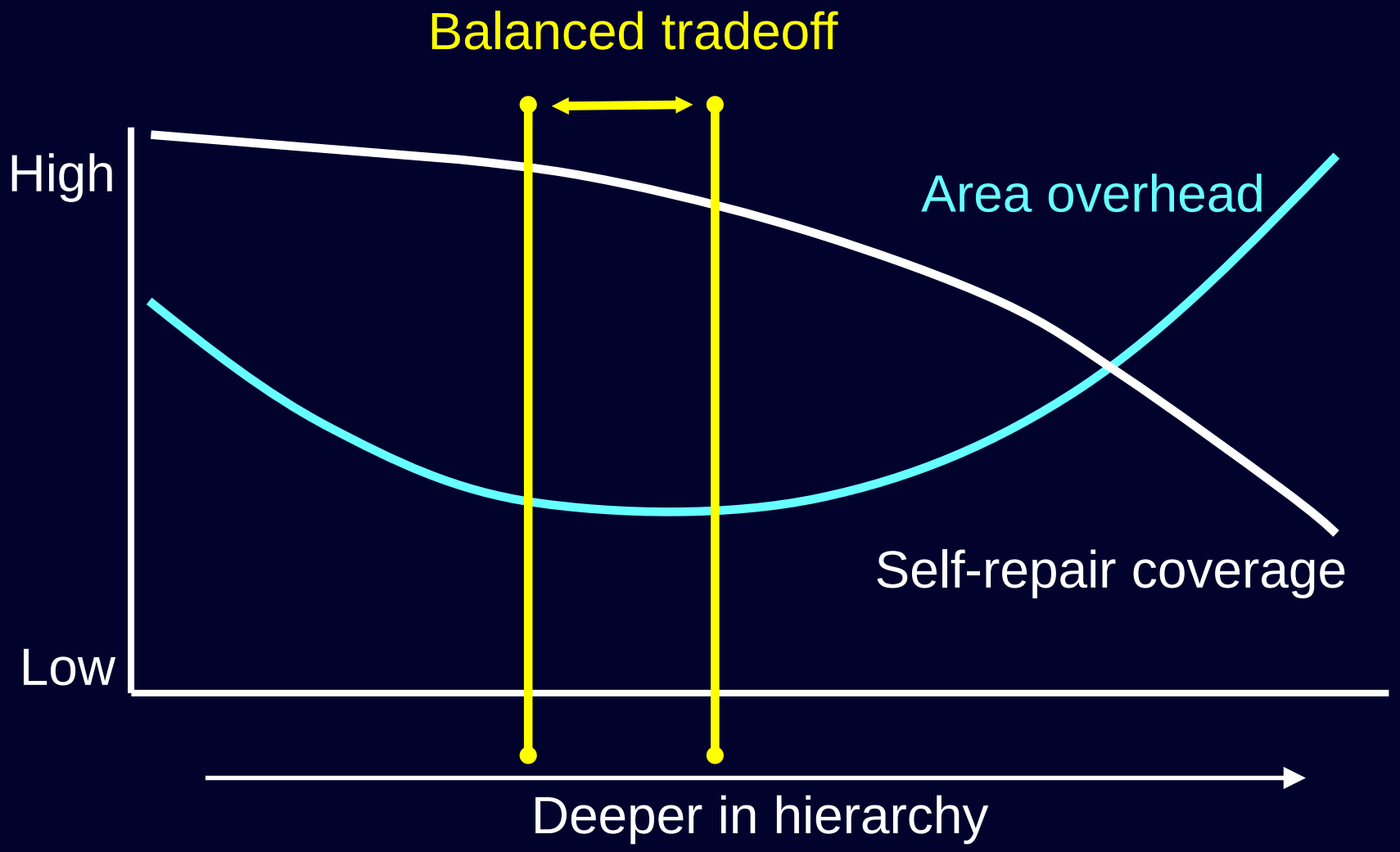
Sparing in Different Levels: Mixed Levels



Sparing in Different Levels: Fine-Grained

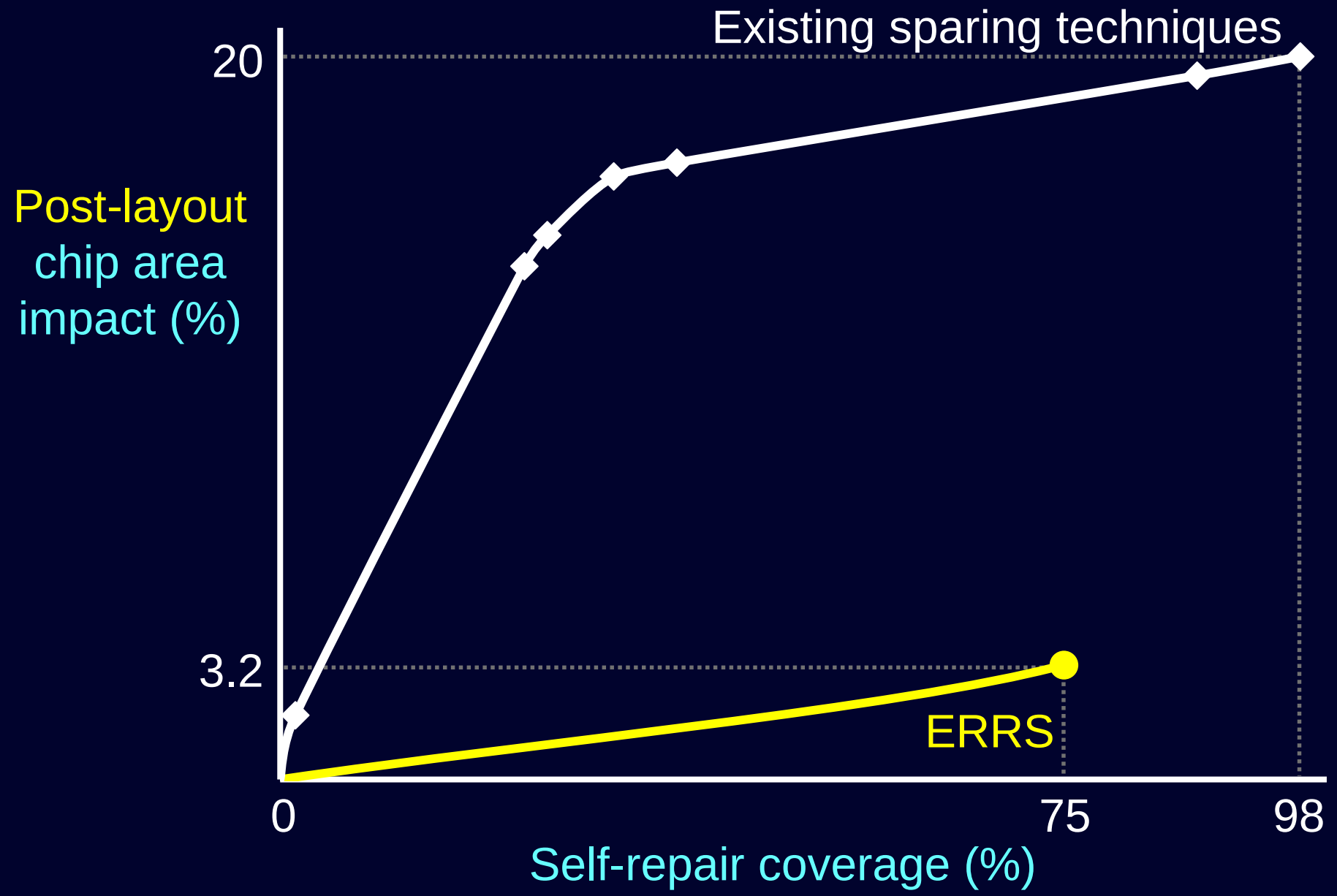


How to Obtain “Sweet-Spot”?

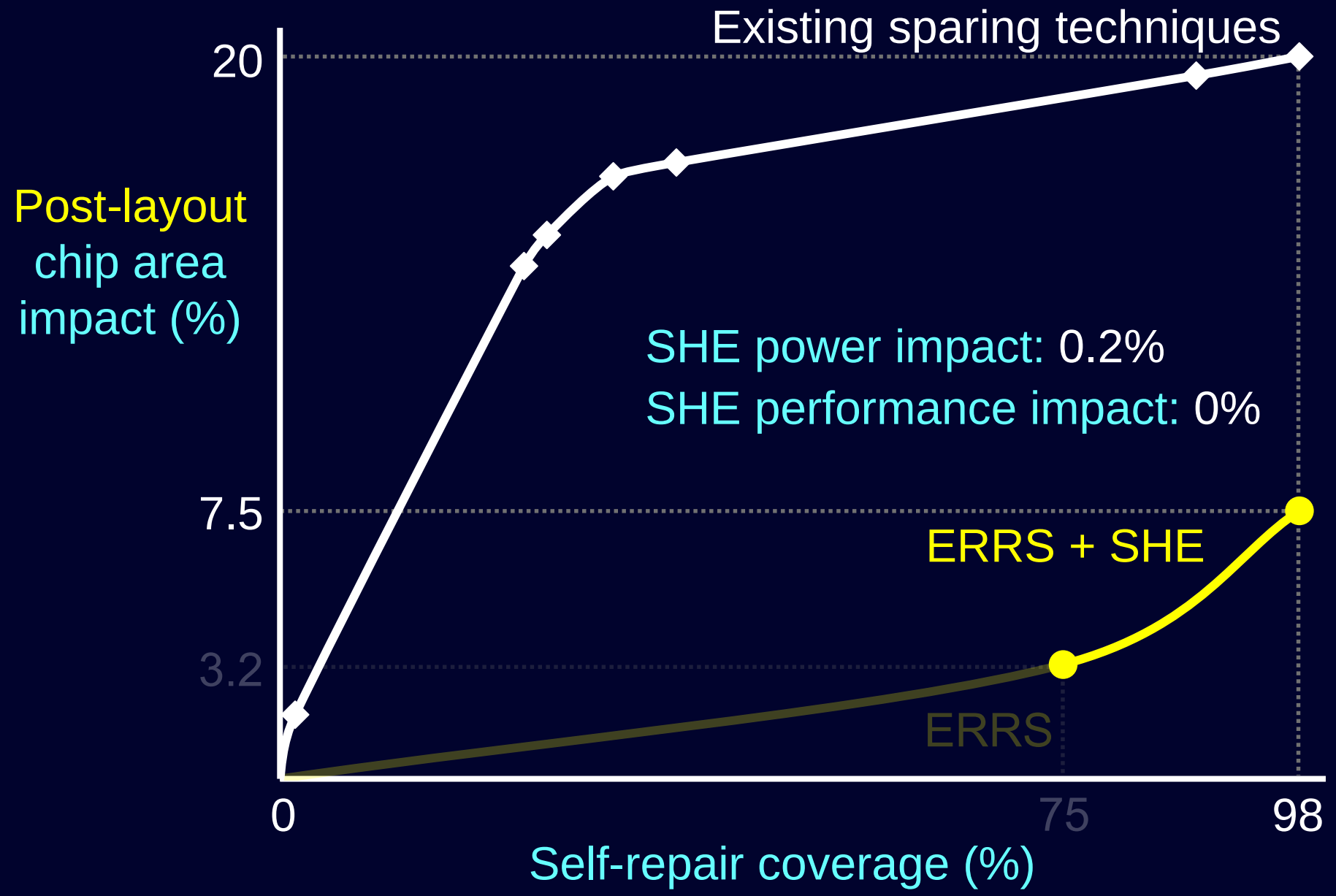


Algorithm details in paper

SHE Results


Existing sparing techniques
20
Post-layout chip area impact (%)
3.2
ERRS
0
75
98
Self-repair coverage (%)

SHE Results


Existing sparing techniques
20
Post-layout chip area impact (%)
SHE power impact: 0.2%
SHE performance impact: 0%
7.5
ERRS + SHE
3.2
ERRS
0
75
98
Self-repair coverage (%)

What About Diagnosis?

- Traditional fault diagnosis difficult
- Effect-cause: infeasible
  - RTL unavailable
- Cause-effect: impractical
  - 7 petabyte fault dictionary
    - [Beckler ITC12]

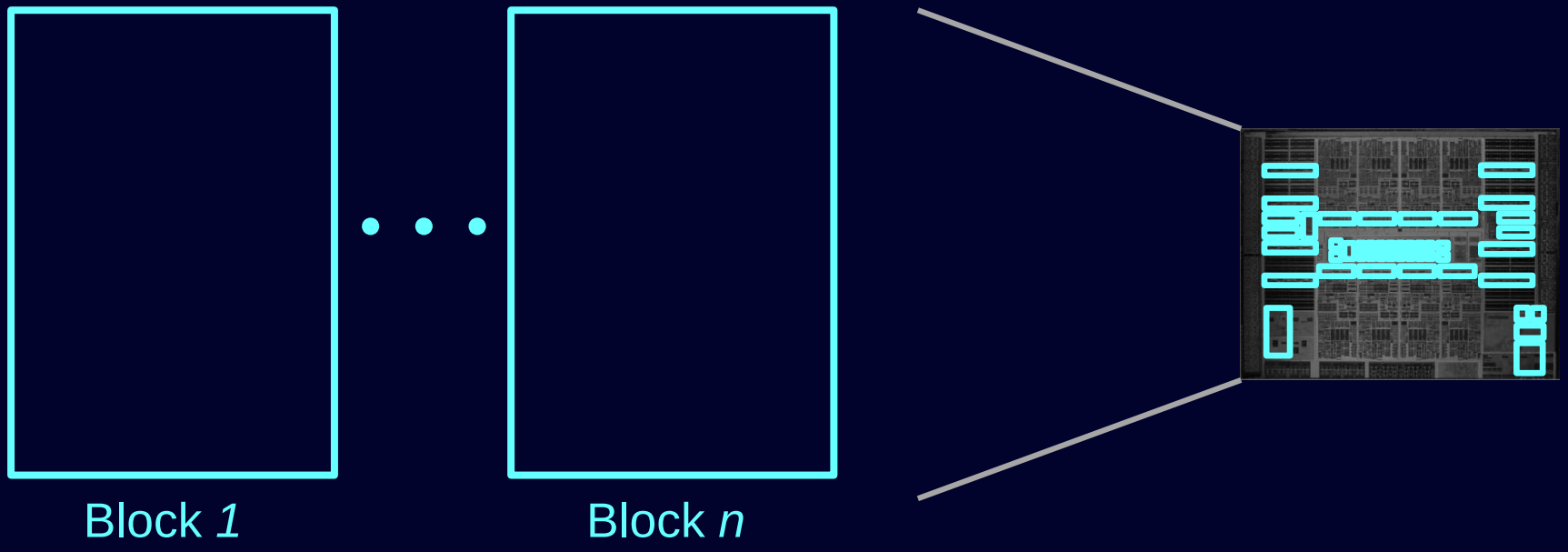
Fault Diagnosis Simplified

- CASP [Li DATE08, VTS10]
  - Concurrent, Autonomous, Stored test Patterns

Fault Diagnosis Simplified

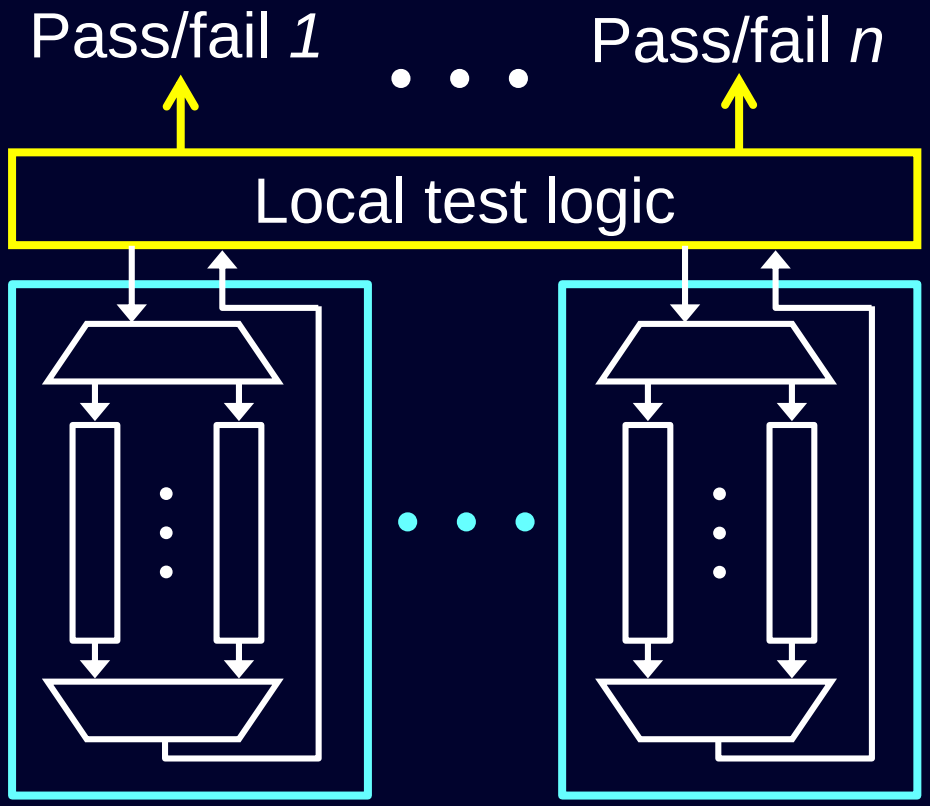
- CASP [Li DATE08, VTS10]
  - ❖ Concurrent, Autonomous, Stored test Patterns

Blocks = self-repair granularity



Fault Diagnosis Simplified

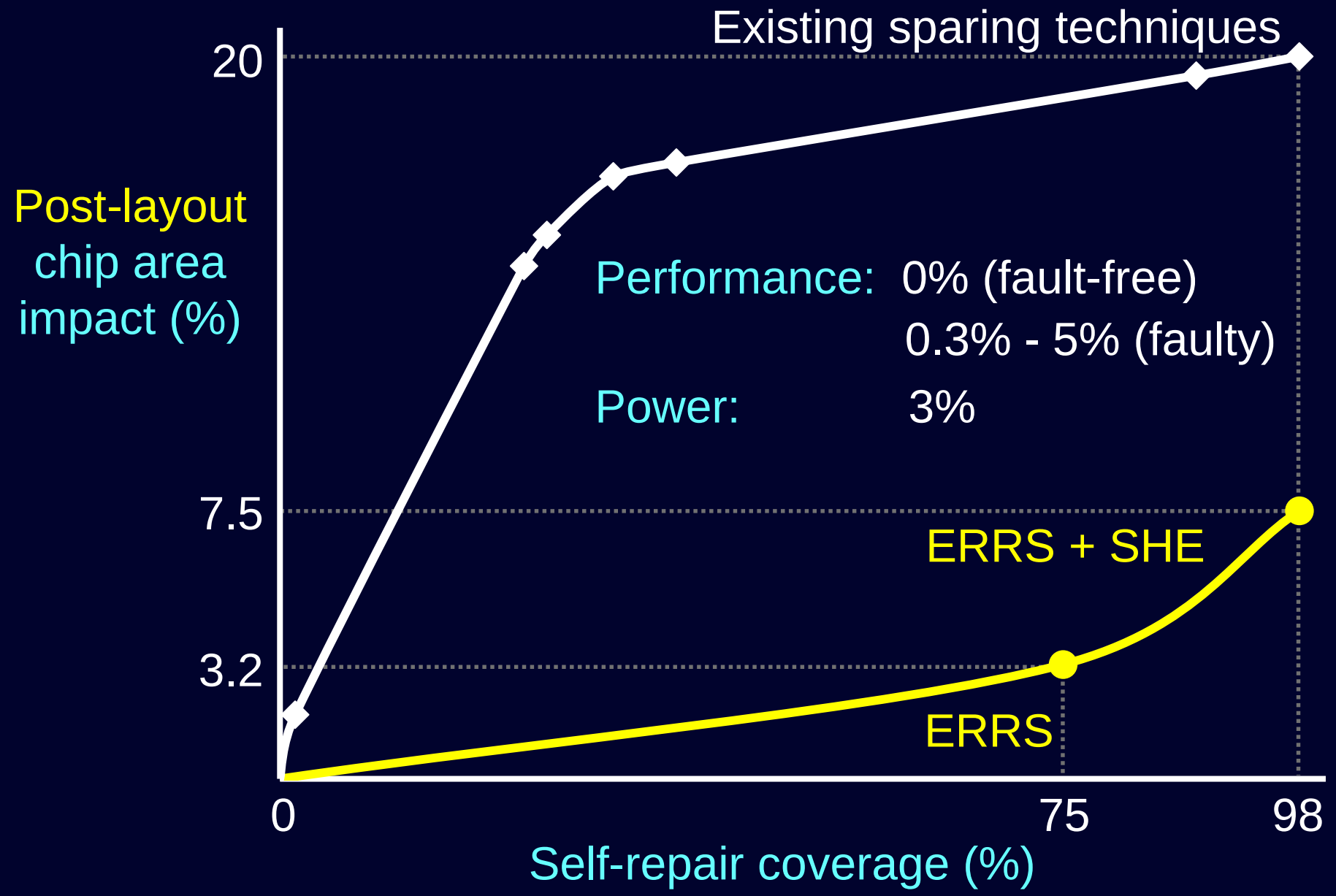
- CASP [Li DATE08, VTS10]
  - ❖ Concurrent, Autonomous, Stored test Patterns



Fail → self-repair needed

Diagnosis = detection

Summary: Self-Repair of Uncore Components



References

[Aitken 04] Aitken, R., “A Modular Wrapper Enabling High Speed BIST and Repair for Small Wide Memories,” *Proc. Intl. Test Conf.,* pp. 997-1005, 2004.

[Chang 07] Chang, J., *et al.*, “The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series,” *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846-852, 2007.

[Kurdahi 06] Kurdahi, F.J., *et al.*, “System-Level SRAM Yield Enhancement,” *Proc. Intl. Symp. Quality Electronic Design,* 2006

References

[Li VTS10] Li, Y., *et al.*, “Concurrent Autonomous Self-Test for Uncore Components in System-on-Chips,” *Proc. VLSI Test Symposium,* 2010.

[Li DATE08] Li, Y., S. Makar, and S. Mitra, “CASP: Concurrent Autonomous Chip Self-Test using Stored Test Patterns,” *Proc. Design, Automation, and Test in Europe,* pp. 885-890, 2008.

[Li ITC 13] Li, Y., *et al.*, “Self-Repair of Uncore Components in Robust System-on-Chips: An OpenSPARC T2 Case Study,” *Proc. IEEE Intl. Test Conf.*, pp. 1-10, 2013.

References

[Powell 09] Powell, M.D., *et al*., “Architectural Core Salvaging in a Multi-Core Processor for Hard-Error Tolerance,” *Proc. Intl. Symp. on Computer Architecture*, pp. 93-104, 2009.

[Romanescu 08] Romanescu, B.F., and D.J. Sorin, “Core Cannibalization Architecture: Improving Lifetime Chip Performance for Multicore Processors in the Presence of Hard Faults,” *Proc. Intl. Conf. on Parallel Architectures and Compilation Techniques*, pp. 43-51, 2008.

References

[Sanda 08] Sanda, P.N, *et al.*, “Fault-Tolerant Design of the IBM Power6 Microprocessor,” *IEEE Micro,* vol. 28, no. 2, pp. 30-38, 2008.

[Schuchman 05] Schuchman, E., and T.N. Vijaykumar, “Rescue: A Microarchitecture for Testability and Defect Tolerance,” *Proc. Intl. Symp. on Computer Architecture,* pp. 160-171, 2005.

[Shirvani 99] Shirvani, P.P., and E.J. McCluskey, “PADded Cache: A New Fault-Tolerance Technique for Cache Memories,” *Proc. VLSI Test Symp.,* pp. 440-445, 1999.

References

[Shivakumar 03] Shivakumar, P., *et al.*, “Exploiting Microarchitectural Redundancy for Defect Tolerance,” *Proc. Intl. Conf. on Computer Design*, pp. 481-488, 2003.

[Sridharan 12] Sridharan, V., *et al.*, “A Study of DRAM Failures in the Field,” *Proc. Intl. Conf. High Performance Computing, Networking, Storage and Analysis,* pp. 1-11, 2012.

[Schuchman 05] Schuchman, E., and T.N. Vijaykumar, “Rescue: A Microarchitecture for Testability and Defect Tolerance,” *Proc. Intl. Symp. on Computer Architecture,* pp. 160-171, 2005.