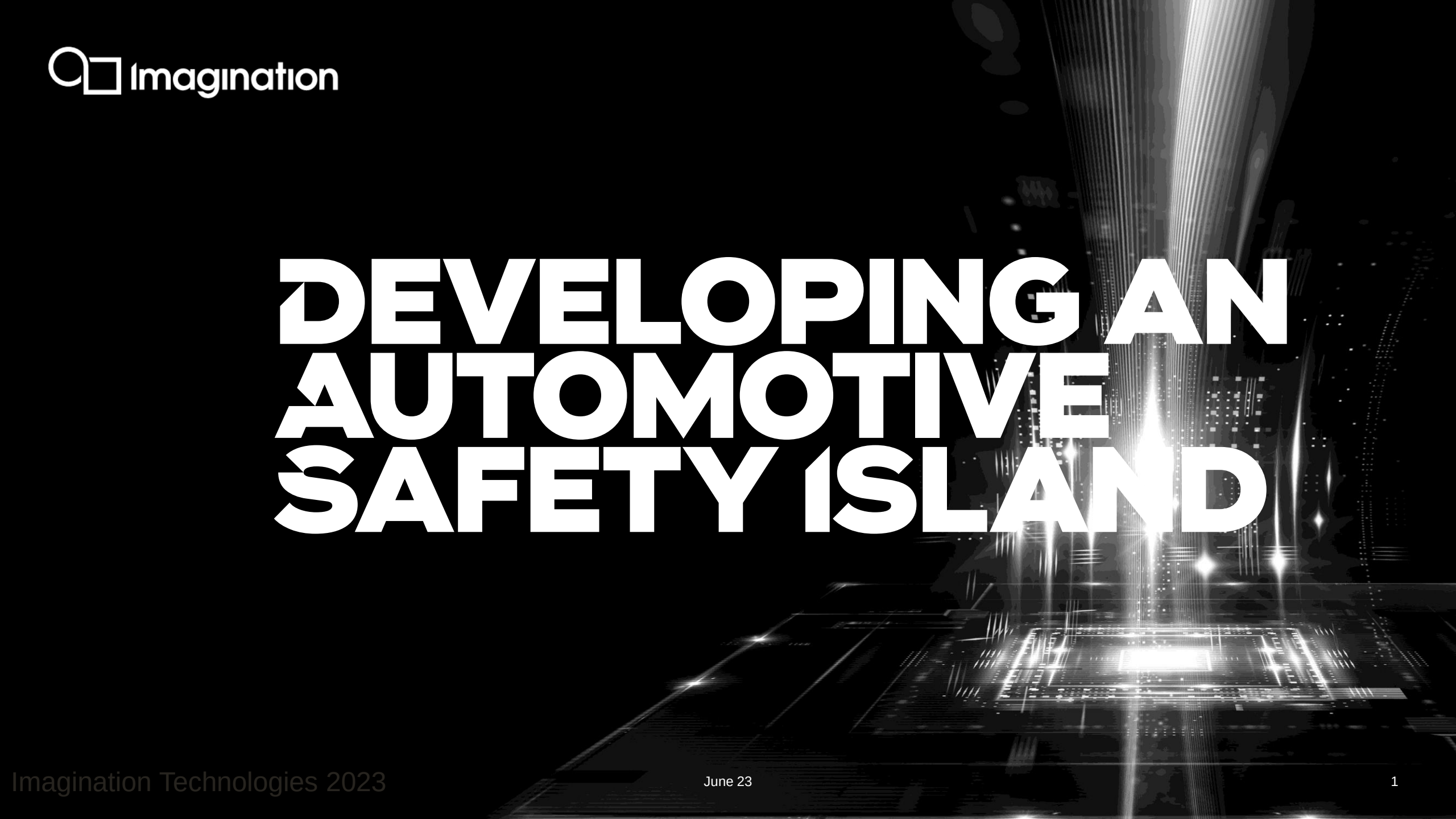
Imagination

DEVELOPING AN AUTOMOTIVE SAFETY ISLAND

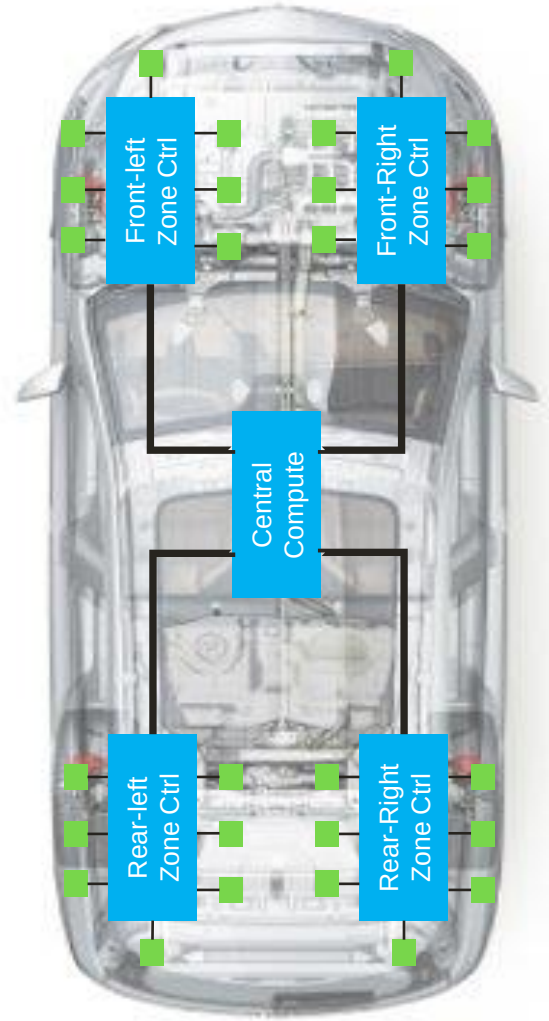


Vehicle Architecture Trends

Electrification	ADAS / Autonomy	Connectivity

Increased compute requirements
Increased software content
Safety remains paramount
Security becoming critical
Consolidation of compute resources
Drive for standardisation

Discrete ECUs → Domain Controllers → Zonal Controllers



Functionally Safe Systems

Safety Integrity Levels

Automotive systems are rated as 1 of 4 “Safety Integrity Levels”

ASIL-A for the lowest level, through ASIL-D for the highest

ASIL-D Costs

ASIL-D requirements increase cost through:

- Rigorous control over development process
- Additional documentation requirements
- Inclusion of redundancy mechanisms
- Exclusion of difficult to analyse technologies
- Reducing sharing of resources

Typical hardware safety features used to achieve ASIL-D

Dual Core Lockstep

To detect errors in logic, a redundant copy of a core processes the same inputs as the functional core (usually with a delay of a few cycles). Outputs are compared and any differences indicate a fault.

Provides excellent coverage, but is expensive and does not offer fault correction.

ECC or Parity

Data (either in memory or in transit on busses) may be protected by ECC (Error Correction Codes).

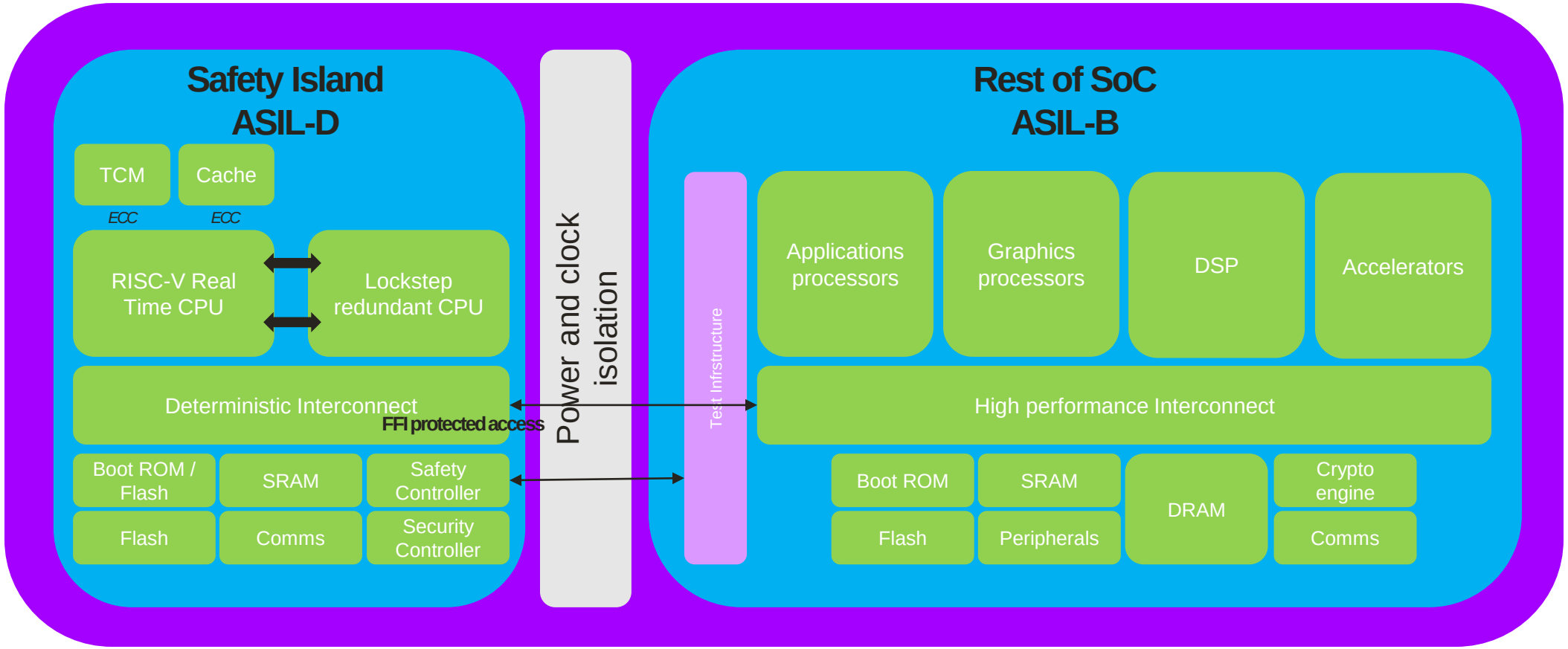
Provides fault correction as well as detection. Requires extra memory and/or routing, and may add delays to critical paths limiting frequency.

Mixed Criticality ECU functionality

- Zonal controllers host a range of functionality in one ECU
  - Some of these functions may require high safety integrity
  - Much of the functionality may require a lower safety integrity level
  - For example, one controller may implement
    - Anti-lock braking, requiring ASIL-D
    - Brake light control, requiring ASIL-B

Design everything to ASIL-D	Separate SoCs for different safety levels	Mix criticality on a single chip
Very expensive Complex functionality can difficult to implement to ASIL-D	Inefficient communication Higher BoM Lower reliability	High criticality functionality needs isolation and Freedom From Interference from rest of SoC

Example Safety Island SoC



Freedom From Interference requires that a failure in the ‘Rest of SoC’ (ASIL-B) must not be able to cause a failure in the Safety Island (ASIL-D)

- **Timing and execution**
  - Execution of an ASIL-B function being blocked must not block an ASIL-D function executing
  - Made easier as only ASIL-D functions run on the Safety Island
  - Safety Island code must not block waiting on an action from ASIL-B software
- **Memory**
  - Memory corrupted by faulty execution on the ASIL-B side must not affect Safety Island software
  - Generally, use separate memories with no access to the Safety Island memory from Rest Of SoC
  - Any shared buffers should be in a constrained area in the Safety Island side
    - If accessibility from Rest of SoC is programmable, must be configured by Safety Island software
- **Exchange of information**
  - Safety Island software must treat any data from the Rest-Of-SoC as unreliable (maybe in shared buffer)
  - Validate integrity, ensure corrupted data does not cause failure

The Safety Island

Characteristics	Physically Isolated (power and clock) from Rest of SoC (to provide protection from common mode failures)
	Keep as simple as possible – less components, easier to analyse, less opportunity for failures
	Real time CPU (Typically TCMs and no MMU)
Functions	General ASIL-D workloads
	Control reset and clocks for Rest of SoC
	Monitor the rest of the SoC for safety failures
	Provide resilient communication to other ECUs
	Coordinate in-service BIST
	Security monitoring

Summary

Industry trends driving move to more compute, and much more software

Architecture moving from separate ECUs, to Domain controllers, to Zonal /Centralised controllers

Increased need to mix safety criticality on a single SoC

Best achieved using a high-safety Island