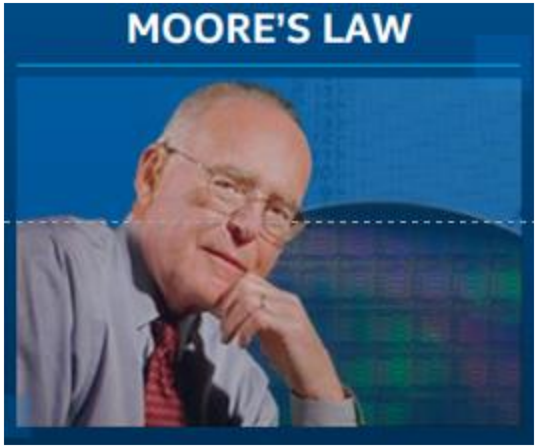
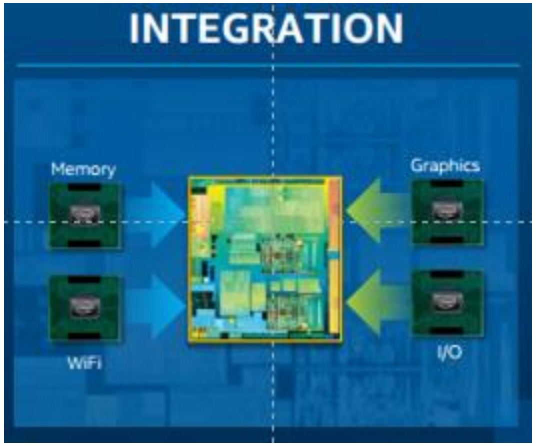
Verification in the age of Integration

John O'Leary
Intel Corporation
john.w.oleary@intel.com

Integration

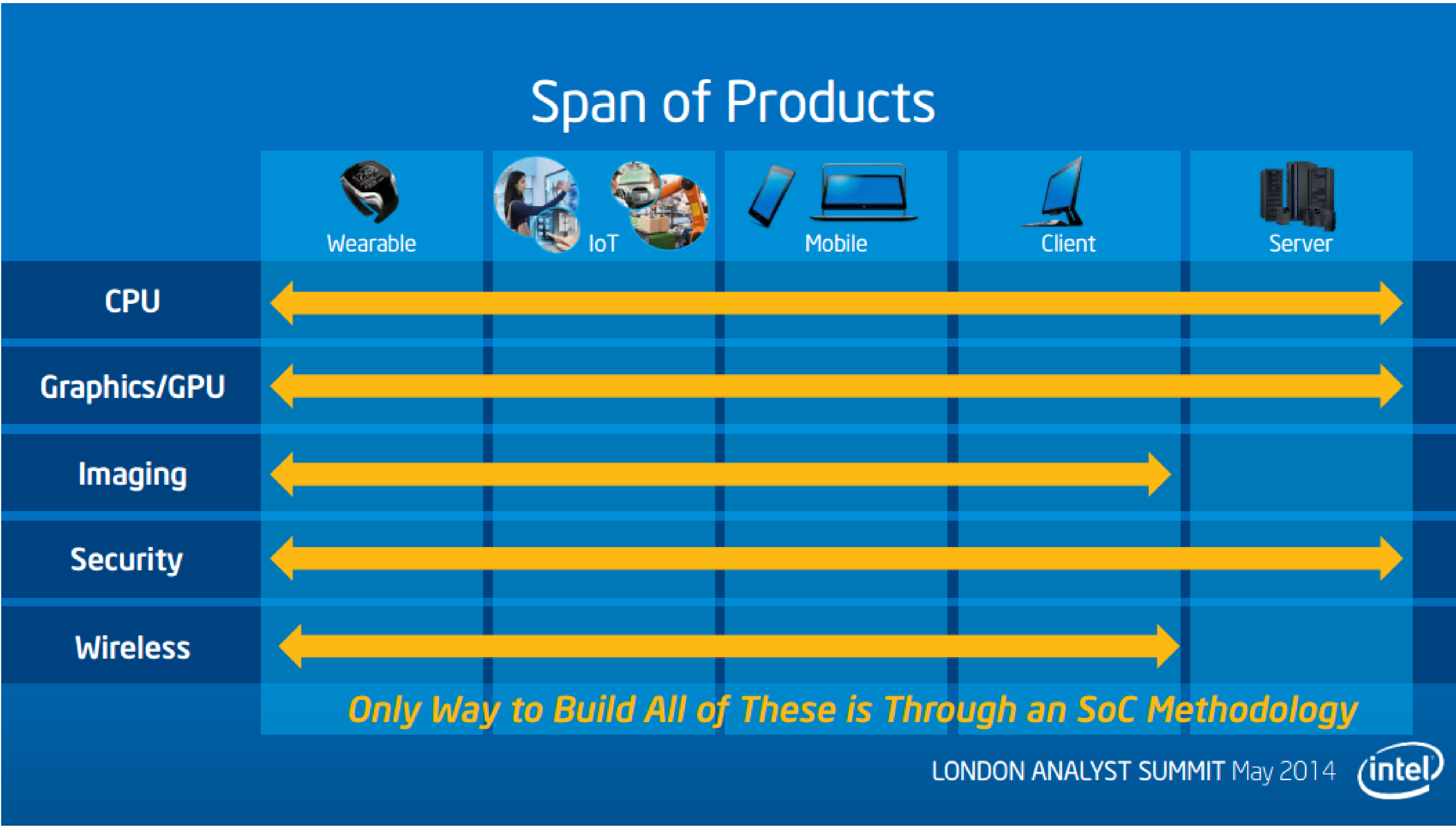




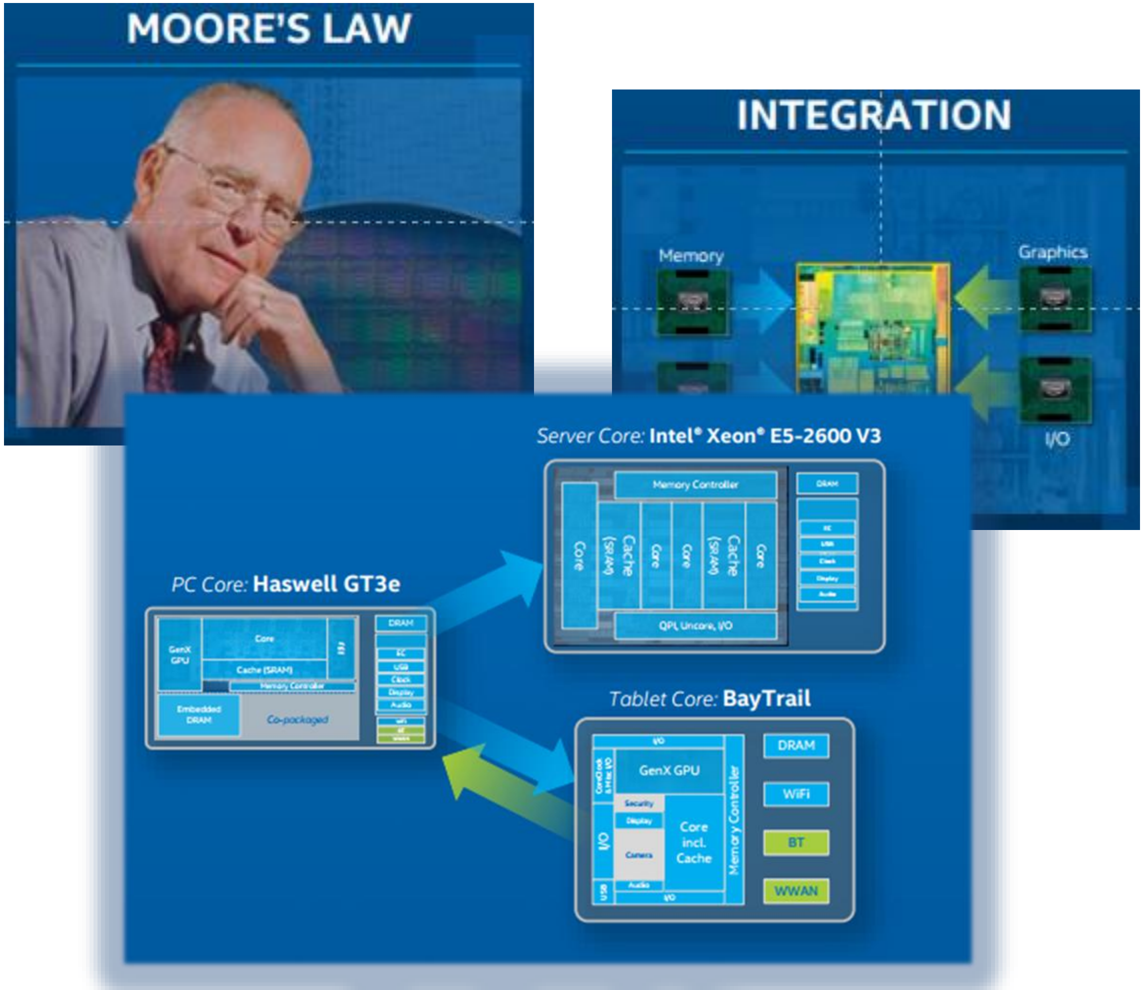
Agenda

- The changing nature of design at Intel
- Easy problems
- Hard problems
- Prospects

Market segmentation

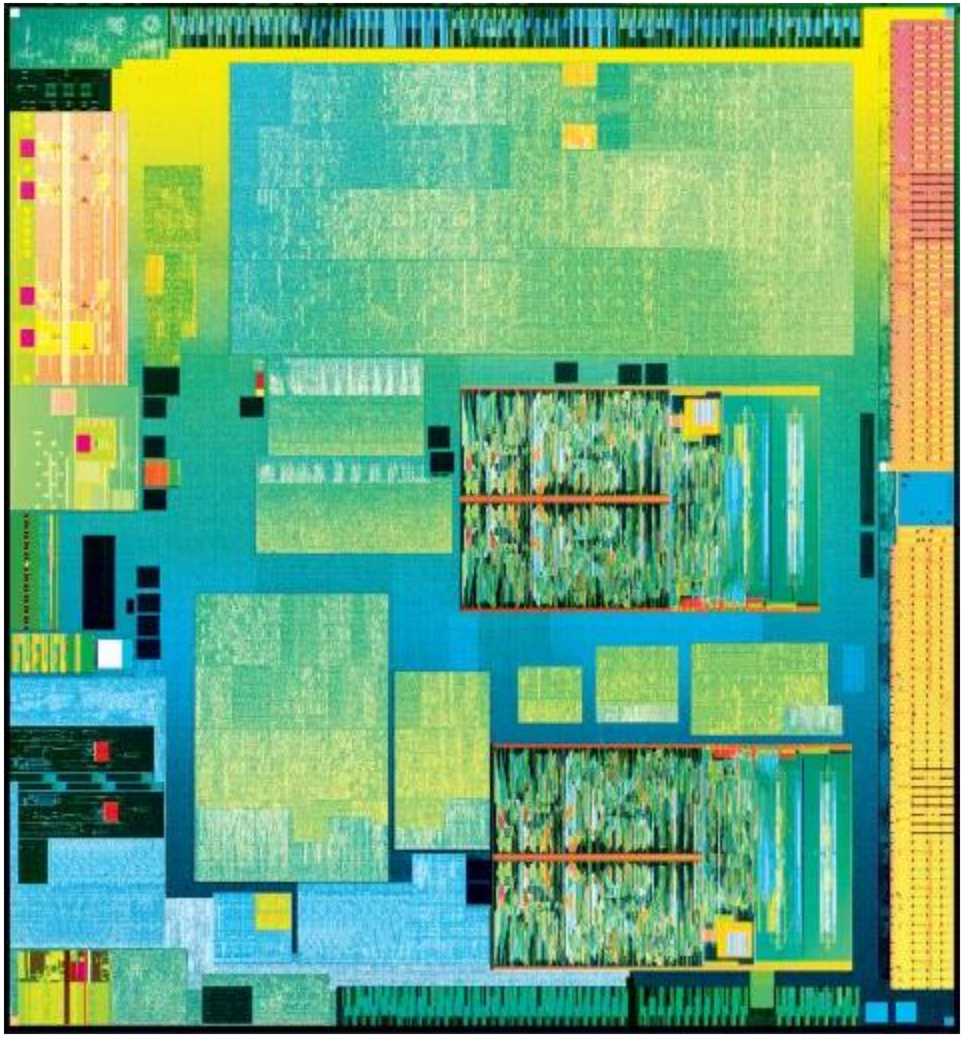


SoC methodology

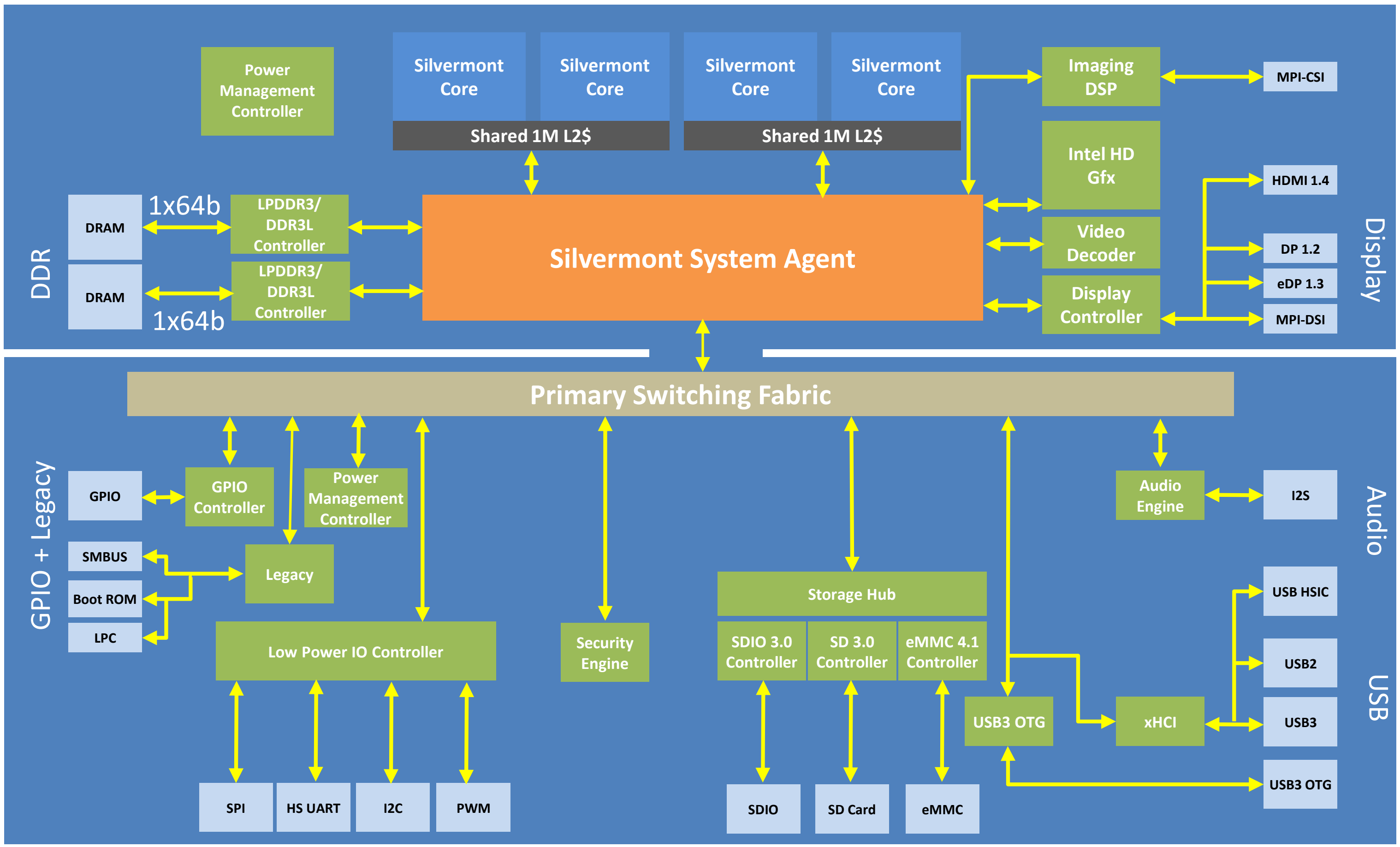


Shared ingredients

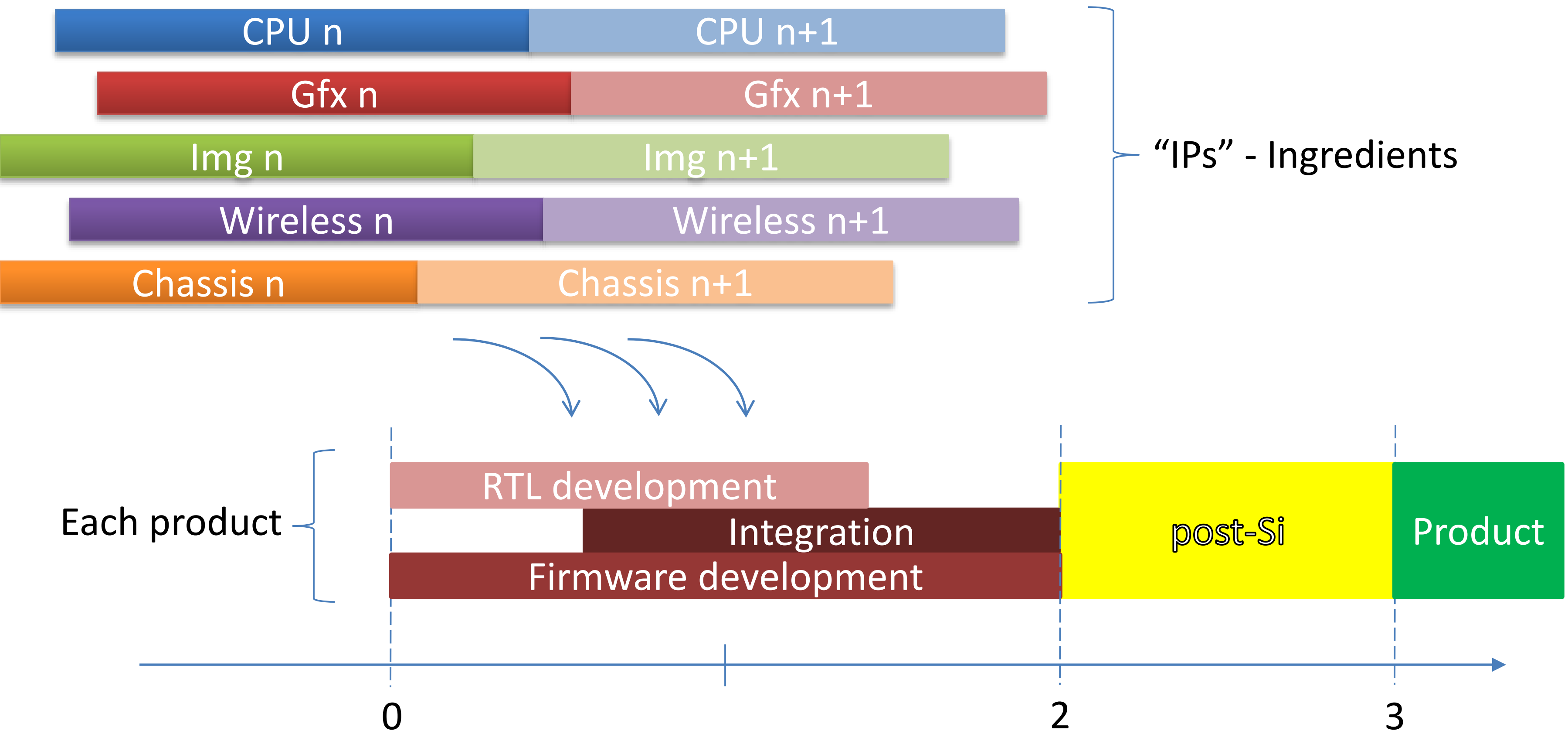
Bay Trail SoC



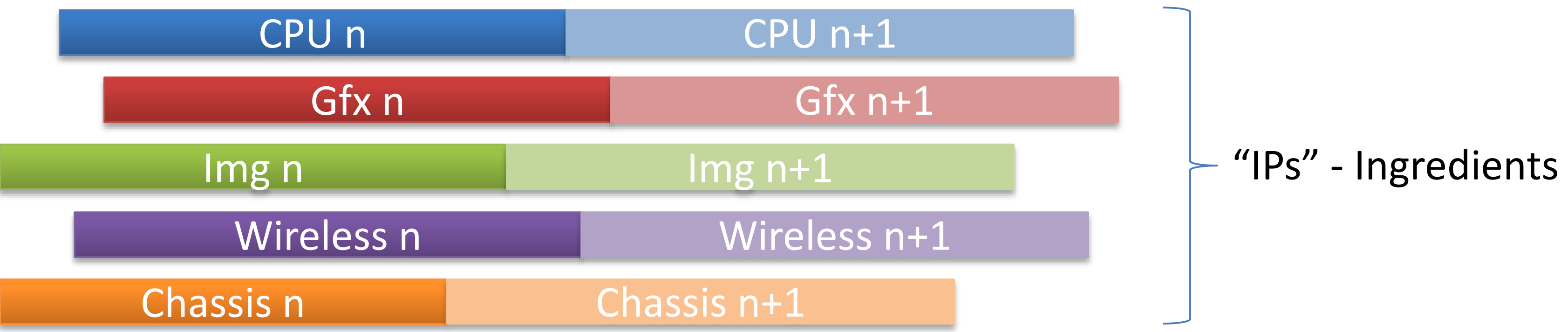
Bay Trail SoC



Development with shared IPs



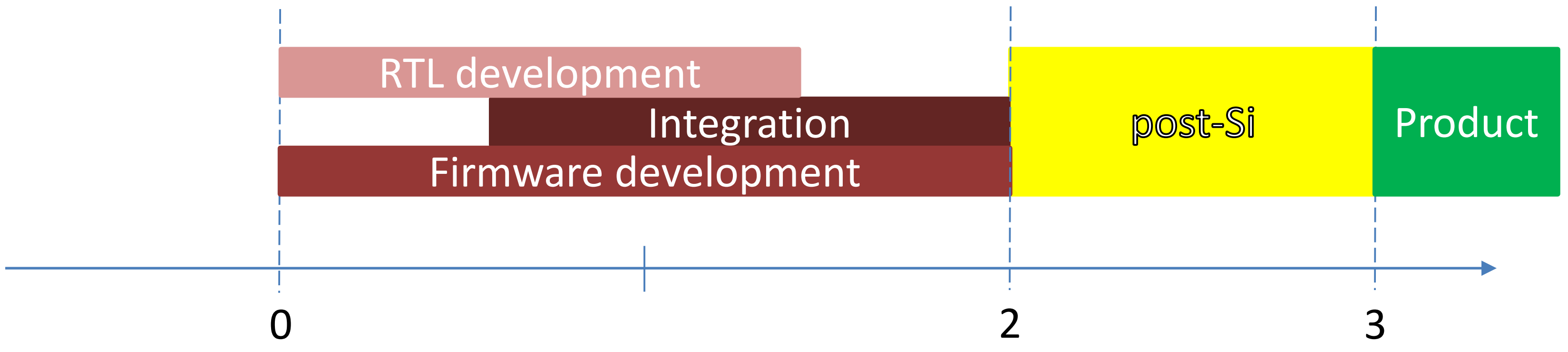
Formal verification today



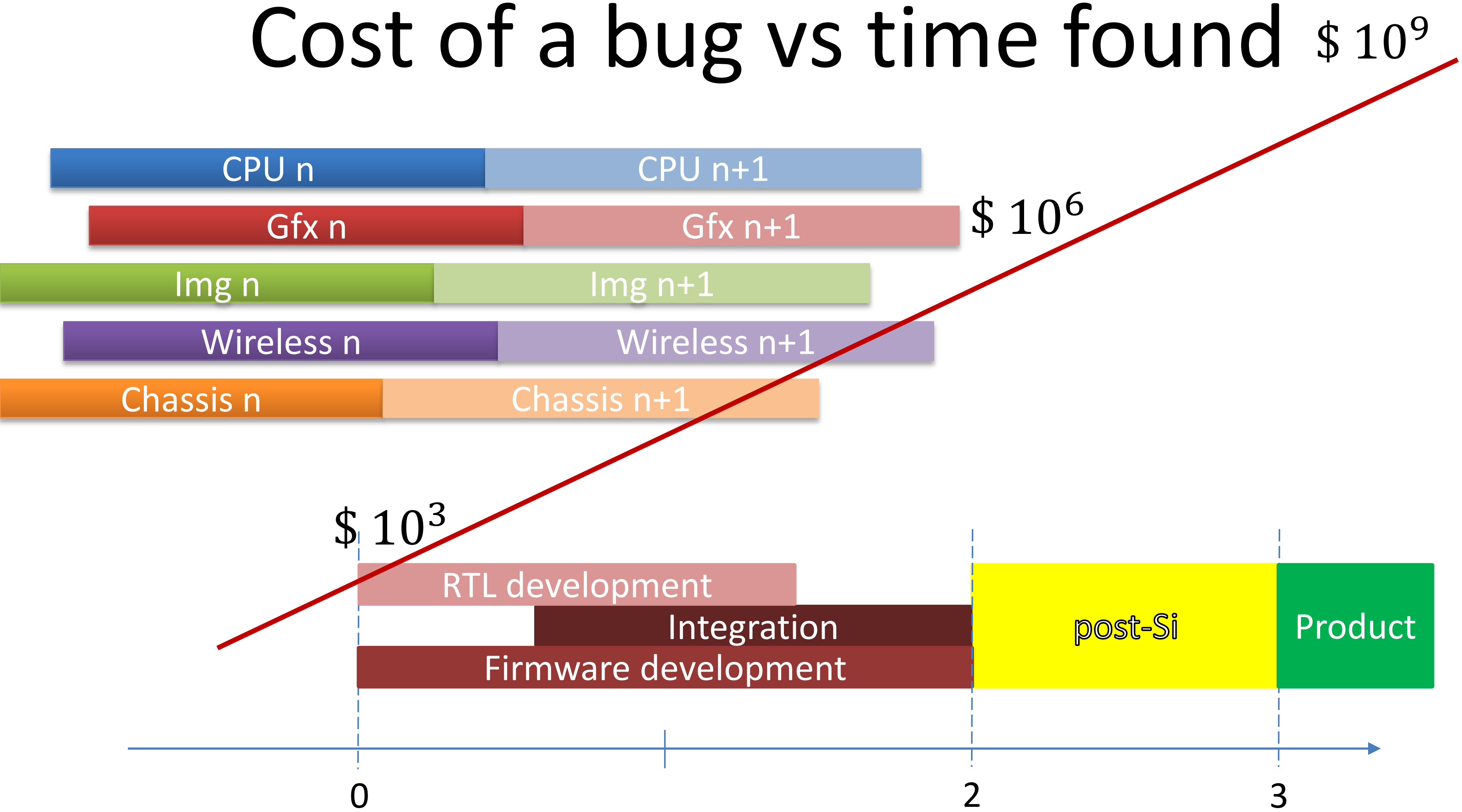
- Primary focus is on units within IPs
  - Does this multiplier multiply?
  - Does this decoder decode?
  - etc

Integration validation

- Simulation and emulation dominate
- Little/no use of formal – but opportunities exist



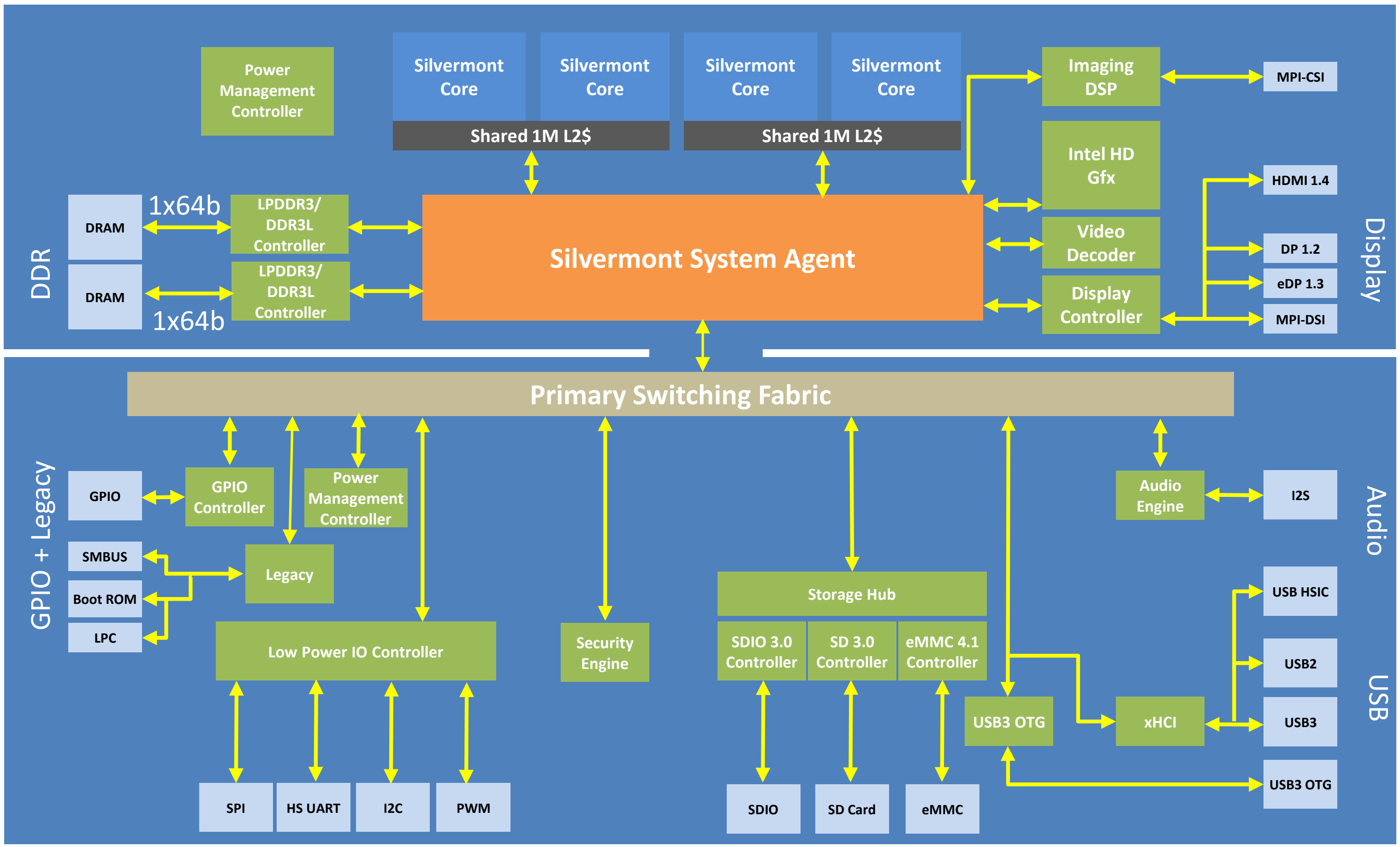
Cost of a bug vs time found



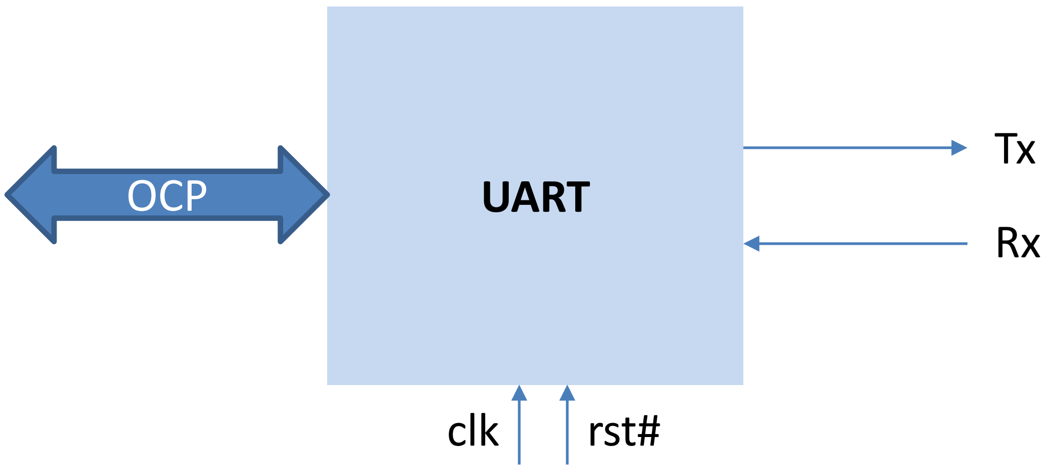
Agenda

- The changing nature of design at Intel
- Easy problems
  - Interface protocol compliance
  - Control/status register (CSR) verification
  - Connectivity verification
- Hard problems
- Prospects

Bay Trail SoC



Interface protocol compliance



- Each IP has several interfaces:
  - Mainband
  - DFx: test, debug, …

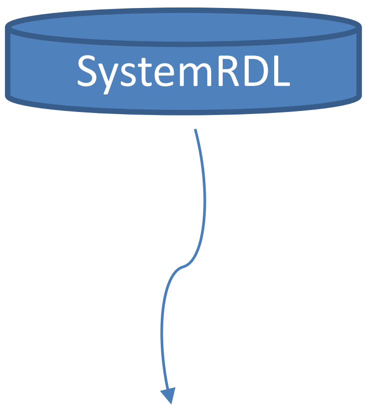
Interface compliance

- Given:
  - IP block
  - Protocol configuration: data width, burst size, feature inclusion, …
- Verify:
  - Interface well-formedness: signals, naming conventions
  - Legality of configuration
  - Adherence to protocol rules

Interface compliance

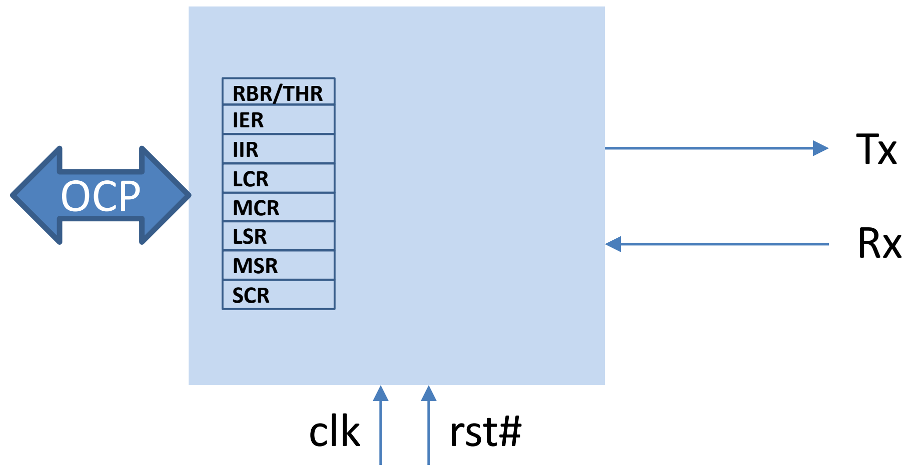
- Standard protocols
  - Commercial bus functional models and checkers support simulation
  - Some EDA vendor support for formal (e.g. Jasper IPKs)
- Proprietary protocols
  - Writing and maintaining high quality formal compliance checkers is very labor intensive
  - Need synthesis of formal compliance checkers (and simulation testbench, …) from declarative protocol specifications

Control/status register verification




```
#define UART_BASE       0x03F8

enum    {
        UART_RBR = UART_BASE + 0,
        UART_THR = UART_BASE + 0,
        UART_IER = UART_BASE + 1,
        UART_IIR = UART_BASE + 2,
        UART_LCR = UART_BASE + 3,
        UART_MCR = UART_BASE + 4,
        UART_LSR = UART_BASE + 5,
        UART_MSR = UART_BASE + 6,
        UART_SCR = UART_BASE + 7,
        };
```



- IPs configured and controlled via CSRs
- CSRs are memory or IO mapped for access by BIOS, driver, …

CSR verification

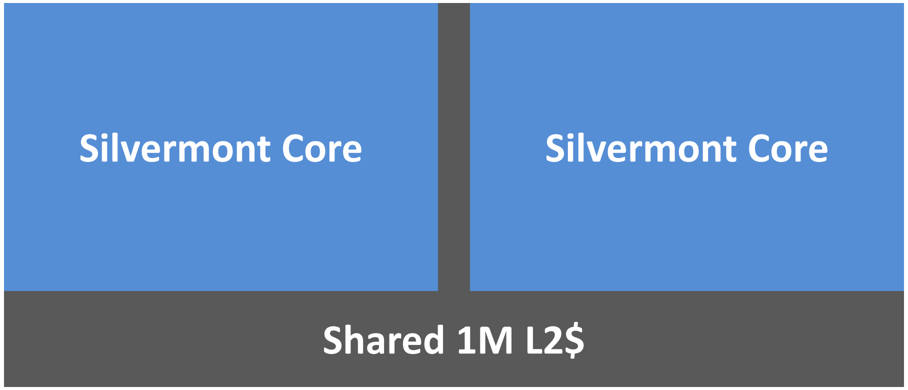
- Given:
  - IP block
  - Register specification (e.g. SystemRDL)
- Verify:
  - Address mapping
  - Correct cold reset values
  - Data integrity
  - Read only/write only
  - Lock bits behavior

CSR verification

- Simulation is standard approach today
- Continuous regressions required due to periodic IP drops and register spec churn
- Formal tools appearing: Cadence Jasper, OneSpin

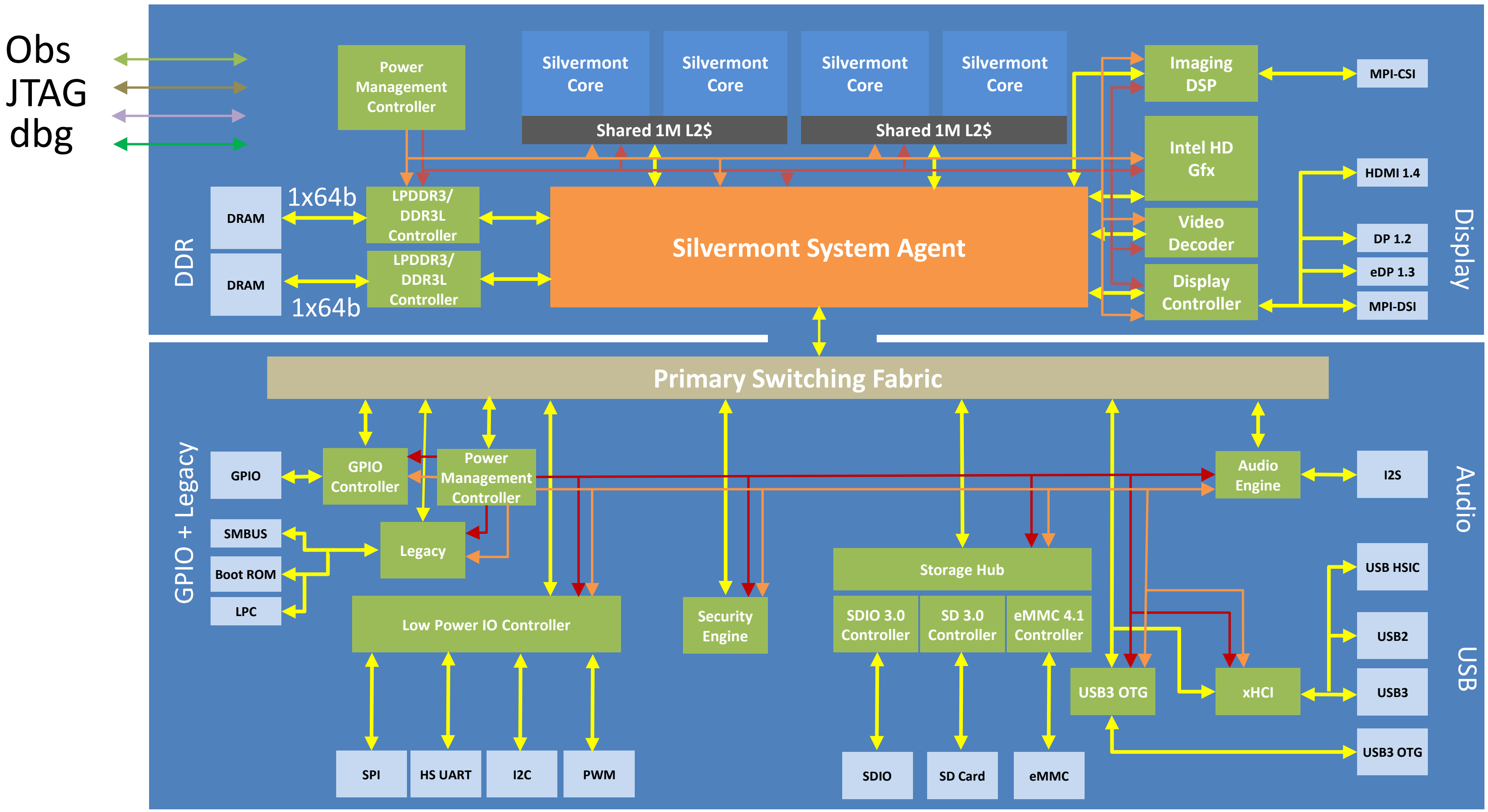
Scaling challenge of “easy” problems

- This is an IP:



- Dozens of interfaces, 1000s of pins
- Complex protocols
- Hundreds/thousands of CSRs, some buried deep

Connectivity verification



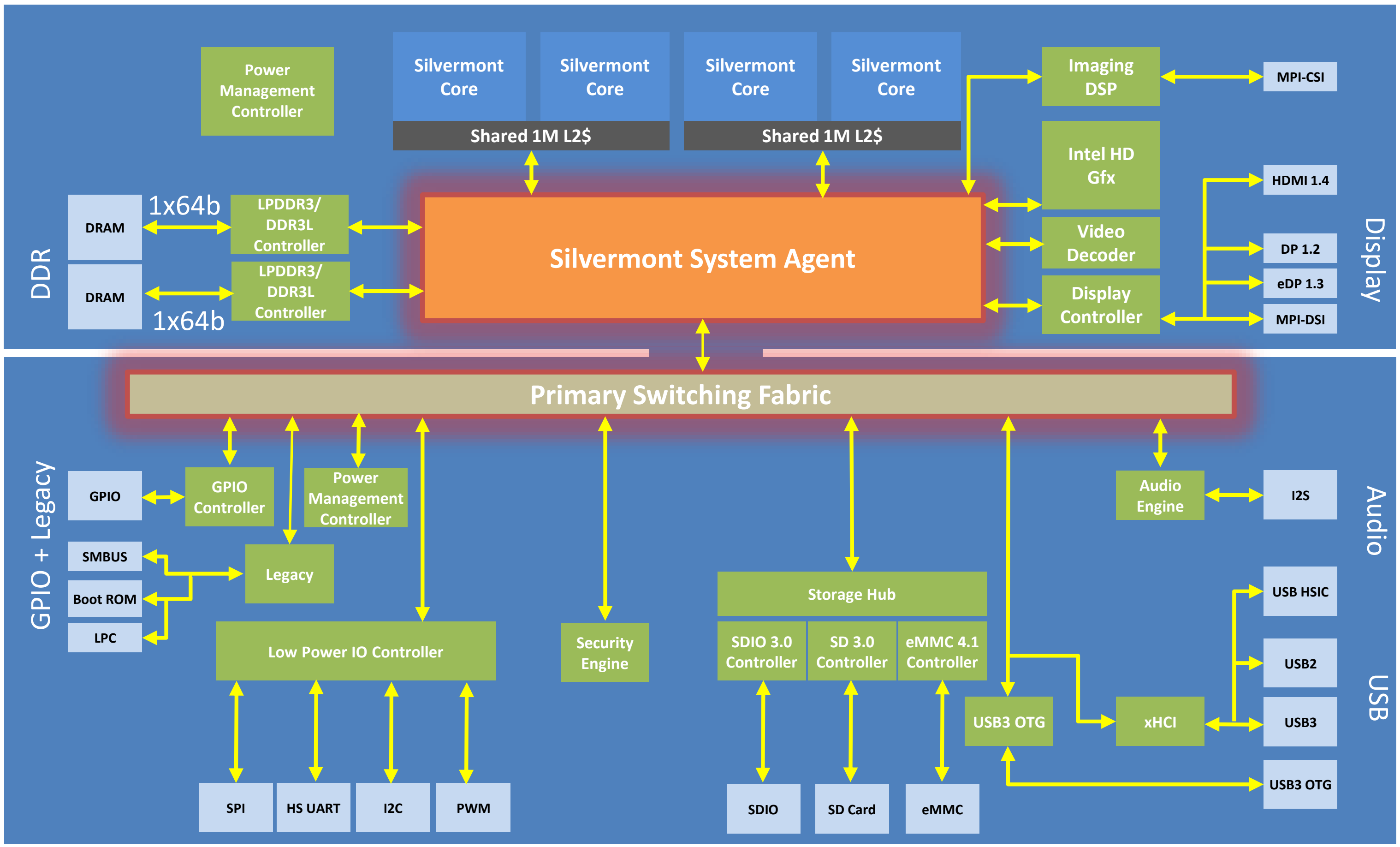
Connectivity verification

- Given
  - Fullchip model including pins
  - Connectivity specification (spreadsheet)
- Verify
  - Power & ground planes connected
  - Clocks & resets connected with correct polarity
  - Fabrics connected to IPs
  - Mux networks correct: GPIOs
  - DFx infrastructure connected: observability, debug, JTAG, misc test, …
- Scaling is the biggest challenge

Agenda

- The changing nature of design at Intel
- Easy problems
- Hard problems
  - Networks on chip
  - Security verification
  - Power management
- Prospects

Networks on chip



NoC verification

- Given
  - NoC RTL
  - Behavior/constraints at network endpoints
  - Maybe: additional info like topology, queue sizes
- Verify
  - Message integrity
  - Deadlock freedom, livelock freedom
  - QoS: latency, throughput

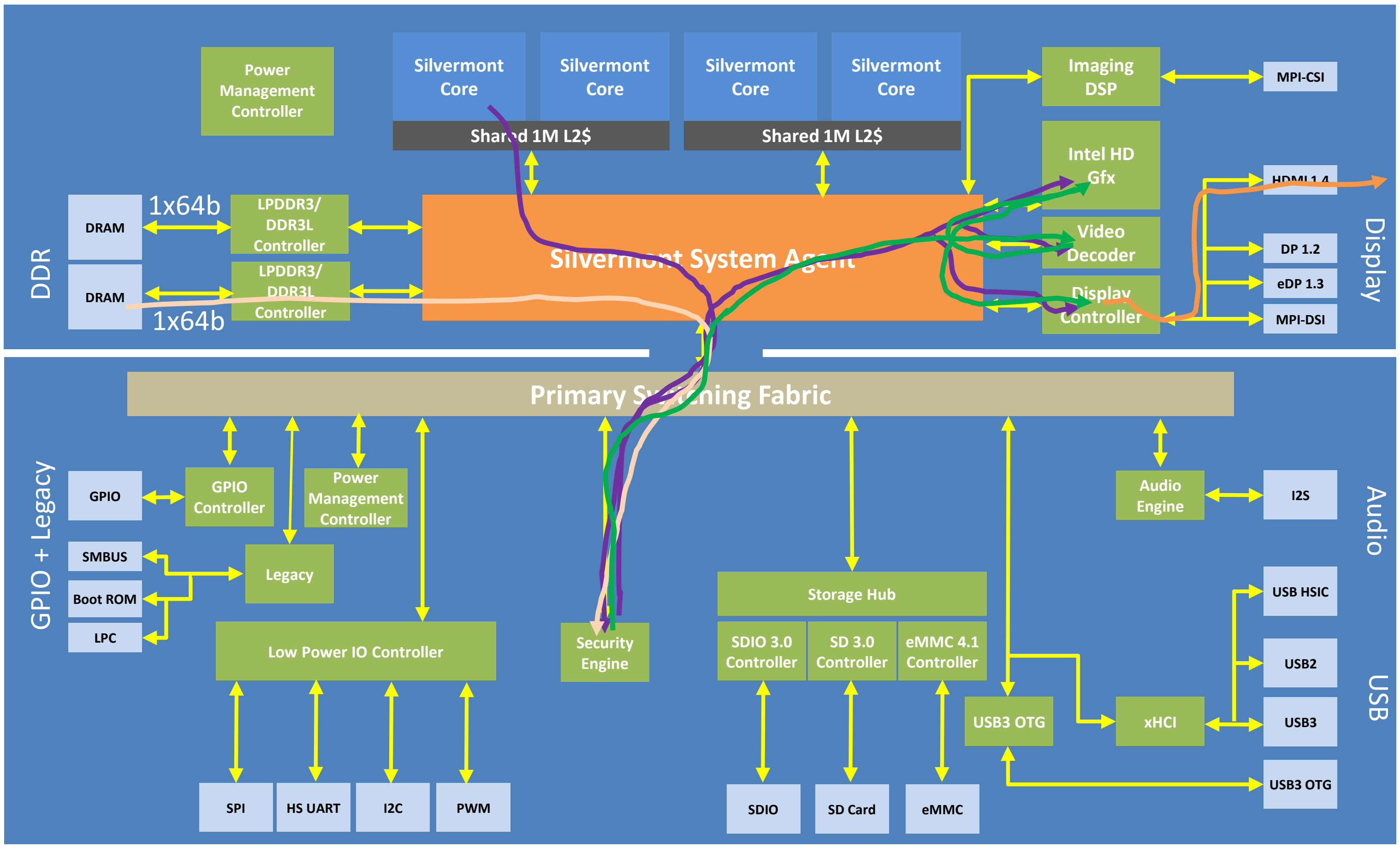
NoC verification

- Usually validated in fullchip simulation or emulation
  - Difficult to hit all important scenarios, corner cases
- Academic traction on formal modeling and verification at architecture level, but no commercial formal tools yet
- Even with verified architecture, establishing RTL correctness extremely difficult

Security verification

- CIA: confidentiality, integrity, availability
  - Focus today is on C and I
- Most effective method is careful manual review by devious experts
- Some formal tools for static security path verification
  - Specify location of secrets and potential attack points
  - Tool seeks to sensitize a path between adversary and secret

Use case: content protection



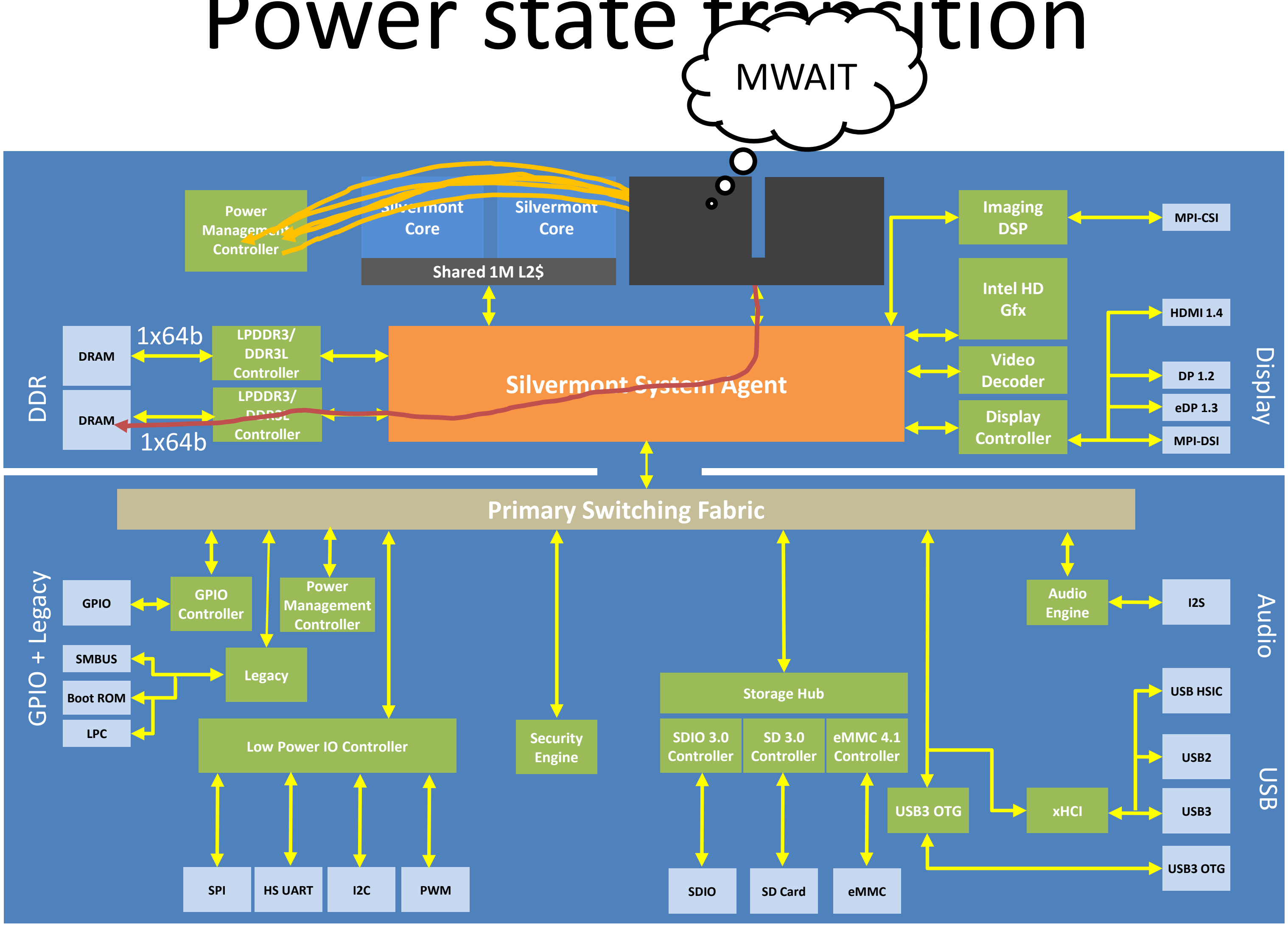
Security verification

- Given:
  - CSR specification
  - IP behaviors
  - Use case flows: content protection, FW authentication, Intel® SGX
  - Threat model
- Verify:
  - C: secret not revealed to adversary
  - I: secret not tampered by adversary
  - [A: secret available to authorized user]
- Analysis must comprehend
  - dynamic nature of the computation,
  - the changing locations of secrets,
  - role and privilege of each participating IP,
  - evolution of access permissions

Power management

- Techniques
  - Clock gating
  - Dynamic voltage and frequency scaling (DVFS)
  - C-states: core active, idle, various levels of powerdown
  - Various system level power states
- Intricate fullchip protocols orchestrate the transitions

Power state transition



Power management

- Verified in emulation or on silicon
- Formal tools to check local power management measures vs power intent (UPF)
  - Clock domain crossings, level shifters
  - Isolation cells
  - Clock and power gating

Power management

- Really need to verify:
  - Power dependencies: e.g. core pup requires fabric pup
  - Flow synchronization: e.g. two cores on one PLL
  - Each IP remains inside its operating envelope
  - Timely response to thermal emergency and other urgent events
  - Stability?
  - Time scales are microseconds or milliseconds

Agenda

- The changing nature of design at Intel
- Easy problems
- Hard problems
  - NoC verification
  - Security verification
  - Power management
- Prospects

Looking ahead

- New verification problems, new opportunities
- Need a verification approach that
  - Supports executable specs and models
  - Enables abstract modeling and refinement
  - Scales to address system complexity
  - Addresses HW, SW, protocols, concurrency

Thank you!